Self Tuning Control of Nonlinear Systems: A Neural Net Based Approach

by

M. M. Farooq Anjum

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

January, 1994

Order Number 1360426

Self tuning control of nonlinear systems: A neural net based approach

Anjum, M. M. Farooq, M.S.

King Fahd University of Petroleum and Minerals (Saudi Arabia), 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

SELF TUNING CONTROL OF NONLINEAR SYSTEMS: A NEURAL NET BASED APPROACH

BY

M.M.FAROOQ ANJUM

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SYSTEMS ENGINEERING

JANUARY 1994

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN, SAUDI ARABIA

This thesis, written by

M.M.Farooq Anjum

under the direction of his Thesis Advisor, and approved by his Thesis committee, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

Thesis Committee:

Md. Shahsir Ahmed
Chairman (Dr. M.S.Ahmed)

L. Cheded
Member(Dr. L.Cheded)

H-Shabaik
Member (Dr. H.Shabaik)

Member (Dr.J.Ezzine)

K. alsultan 3-1-94
Dr. K.S.Al-Sultan
Department Chairman

Dr. Ala H. Rabeh
Dean, College of Graduate Studies

Date: 4.1.94

SELF TUNING CONTROL OF NONLINEAR SYSTEMS:A NEURAL NET BASED APPROACH

M.M.FAROOQ ANJUM

SYSTEMS ENGINEERING

1993

Dedicated to

My Parents,

and Sisters

ACKNOWLEDGEMENT

Praise be to the Lord of the World, the Almighty for having guided me at every stage of my life.

Words are not adequate to describe my gratitude towards my advisor Dr. M.S. Ahmed. It was a real pleasure working with him. I would also like to place on record my appreciation for the cooperation and guidance extended by my committee members, Dr. L. Cheded, Dr. H. Shabaik and Dr. J. Ezzine.

I also acknowledge the generous help and support for this research given by KFUPM. My thanks also to the chairman, faculty and my friends, both from within and outside the department, who made my stay at KFUPM a pleasant and memorable one.

Contents

List of Figures

Abstract

Name: M.M.Farooq Anjum

Title: Self tuning control of nonlinear systems :
A neural net based approach

Major Field: Systems Engineering

Date of Degree: 1993

In recent years there has been considerable interest in the area of artificial neural networks. One of the application areas of neural networks is in adaptive control systems. In this work, use of neural networks for direct self tuning control of deterministic and stochastic nonlinear systems has been proposed and investigated using simulation study. Minimum variance and generalized minimum variance control schemes have been used. Methods to overcome the slow convergence problem associated with the backpropagation training algorithm have also been proposed and implemented.

Master of Science Degree
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
1993

Nomenclature

α_i, β_i	parameters of the plant
y	actual plant output
u	actual plant input
ϕ	pseudo-output
y_d	desired output
d	delay in the plant
λ	weighting placed on the input
ξ	noise
Ω	backpropagated error
e	error
η	training rate coefficient
J	cost function
$F(.)$	activation function
Γ	gradient of the neural network

خلاصة الرسالة

اسم الطالب : محمد فاروق أنجم
عنوان الرسالة : تحكم التوجية الذاتي للأنظمة الغير خطية: طريقة تعتمد على الشبكات العصبية.
التخصص: هندسة النظم والأساليب.
تاريخ التخرج : رجب ١٤١٤هـ.

ظهر اهتمام ملحوظ في السنوات الأخيرة بالشبكات العصبية الصناعية. وتعد أنظمة التحكم ذات التوجيه الذاتي واحدة من التطبيقات لها. في هذه الدراسة، تم استخدامها في التحكم المباشر الذاتي التوجية للأنظمة الغير خطية وتم استخدام المحاكاة لدراستها. كما تم استخدام طريقة التحكم بالمتغير الأدنى والمتغير العام. كما تم اقتراح وتطبيق طرق لتلافي مشاكل بطء الوصول للحل المتعلقة بخوارزميات تدريب الانتشار العكسي.

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران-المملكة العربية السعودية
رجب ١٤١٤هـ

Chapter 1

INTRODUCTION

Summary: A brief look at the motivation behind the research in the field of control systems is taken. The motivation for the use of neural networks as controllers is also established. Finally the organization of the thesis is presented.

1.1 Motivation

The float valve regulators and water clocks of antiquity, James Watt's centrifugal governor, the radar antennas or the autopilots are all proof of the importance of control systems to mankind from ages past. These are but a few examples of the numerous successful applications and implementations of conventional control wherein the goals of a controller are fixed and defined by the designer. Conventional control theory is perspective. It depends on a model of the system, crisp logic as well as relational representation.

With life getting more complex and more demands being placed on control engineers, the principles of conventional control prove to be insufficient to control many

real life situations that require less stringent assumptions.

As control methods began to find their way into practical applications, they opened a Pandora's box to a wide spectrum of complex applications. The distinctive features of these complex systems include poor models, high dimensionality of the decision space, distributed sensors and decision makers, hierarchies, presence of noise, multiple time scales, various performance criteria, complex information patterns as well as extremely large amounts of data and rigid demands on the desired performance. The difficulties that arise in complex systems can be broadly classified into three categories [1]. The first is computational complexity, the second is the presence of nonlinear relationship with many degrees of freedom and the third is the uncertainty which includes the model uncertainty, parametric uncertainty as well as the presence of disturbances and noise. These difficulties are also responsible for the continued evolution in the field of control. For example a few years back, implementation of a self tuning controller would have been considered computationally infeasible. But now the availability of microprocessor technology has provided an impetus for research into self tuning control. Also, today the theory of nonlinear systems isn't well developed. This is spurring on research in this field.

Building systems with some autonomy means that these systems should be capable of setting their own goals, depending on the constraints and objectives imposed by the higher levels of the hierarchy, present situation and also the background knowledge or experience gained. Such systems may often be required to reason under uncertainty thus involving a great deal of judgmental imprecision. One approach here would be to mimic human decision making. But this is easier said than done. The lack of understanding of this " human-like " intelligence means that current

control techniques do not have the ability to learn or to take decisions under unstructured environments. Hence the need arises for intelligent controllers. These controllers should be capable of decision making under uncertainty. They should also possess a highly decentralized behavior to facilitate fast and simple processing.

These requirements posed by the increasingly challenging problems of modern society caused the control system community to look around for better "pastures". A natural step in this evolution was the use of neural networks in control. Neural networks offer much hope as a tool for on-line learning, optimization and optimal policy making. Their remarkable learning capability is what sets them apart from other modelling techniques.

The desire to mimic the superior human recognition capabilities, which are still beyond human understanding, was the main driving force behind researchers that led them to ultimately develop a new paradigm for learning called Neural networks. Hence neural networks can be described as a human attempt to create an artificial brain – shades of science fiction. But it is to be pointed out that although neural networks keep being improved in their performance, they still remain no match to the human nervous system.

Neural networks also have the capability of massive parallel processing in contrast to the sequential execution of the conventional Von Neumann digital computers. Neural networks also provide, in principle, significant fault tolerance, since damage to a few links does not impair the overall system performance.

Currently the hardware implementation of neural networks is also a very active research area. Some manufacturers, such as Intel, have produced a neural net chip which is supposed to have more effective throughput (for the neural net calculations)

than even the Cray supercomputer. Another company, Oxford Computer in Oxford, Connecticut, has developed an intelligent memory chip that can be used for neural network implementation.

An area of control systems which offers much scope for the use of neural networks is adaptive control. Here the controller has to adapt itself with change in the process parameters. A class of adaptive control algorithm is called self-tuning. A self-tuning algorithm is one in which the control signal generated from estimated parameters turns out to be asymptotically the same control law generated by feedback, if the process parameters were known.

In this work we have concentrated upon methods of utilizing neural networks as self-tuning controllers.A strategy for the minimum variance control of linear and nonlinear systems using neural networks is presented. Schemes for the control of systems with nonmininum phase behavior, which is the rule rather than the exception in the case of discrete time systems, have also been provided. Another important area upon which our attention was focussed was the reduction of computational complexity and the improvement in convergence rate of the neural networks. Extensive simulations on linear as well as non-linear systems have also been performed in order to investigate the feasibility and performance of the proposed methods.

1.2 Organization of the work

Chapter 2 deals with the literature review of the field of neural networks as self tuning controllers. Chapter 3 provides a brief review of the structure and training of neural networks. This chapter also includes the basis of self-tuning control. Chapter 4 is devoted to the development of self tuning algorithms using neural net-

works. The simulation results are all provided in Chapter 5. Chapter 6 includes the contributions, conclusions as well as suggestions for future research.

Chapter 2

LITERATURE SURVEY

SUMMARY: A survey of the literature on self tuning control is provided. In this chapter we also look at the recent trends in the utilization of neural networks for the purpose of adaptive control. An in-depth survey of the literature available on using neural networks as self tuning controllers is provided.

Neural networks for control systems is a rapidly expanding field. The tremendous interest in the field of artificial neural networks can be inferred from the growth of papers, journals, conferences and conference sessions devoted to it. Neural networks can be used in many different application areas with control systems being one of them.

With the interest aroused in applications of neural networks in control systems, there have been quite a few papers on this topic in various journals. A survey of these indicate that some of the major topics for further research include

- Utilization of neural networks for identification and control of nonlinear systems.

- Improving upon the direct neural-network based self-tuning methods for non-linear plants

- Improving the learning and adaptation rate of neural networks.

The self tuning regulator was probably first introduced in [2]. An algorithm for single-input single-output linear systems (SISO) consisting of a least squares estimator and a linear controller has been proposed. However, not much work had been done in this area till the beginning of the seventies. The potential use of the recursive estimation algorithms in the field of adaptive control stimulated further interest in them. The advent of process computers and microprocessors provided added stimulus to the study of the synthesis of controllers and the implementation of recursive parameter estimation techniques.

Astrom [3] has given a complete discussion of a self tuning regulator for SISO systems. Astrom and Wittenmark [4] brought self-tuning control into the stochastic environment. Their method is based on a minimum variance strategy. The main conclusion is that, if the algorithm converges at all, then it converges to the desired minimum variance controller.

Basic self tuning regulators are designed for a situation where the control problem can be characterized as a minimum variance regulation. Hence they cannot be applied to nonminimum phase systems as the control input becomes unbounded in this case. Later Astrom and Wittenmark [5] developed a self tuner having the same approach, but based on pole/zero placement. Several algorithms both implicit and explicit, have been proposed by them.

The idea of self tuning for refinement of existing asymptotic properties is extended by Wellstead [6] so that the designer could make arbitrary assumptions con-

cerning disturbance dynamics. An implicit pole assignment regulator which could be easily formulated from extended self tuner is also reported.

Self tuning controller that attempts to minimize the output variance is highly sensitive to the position of the zeros in the discrete time. In order to overcome these limitations a new class of control laws were proposed in [7], [8], [9], [10], where instead of minimizing the variance of the output, the variance of a pseudo output is minimized.

There have been attempts in recent years to treat self-tuning and model reference adaptive control in a unified way [11]. The practical features that are required in a self tuning controller are discussed in [12]. The occurrence of the discrete-time non-minimum phase zeros due to the plant dead time or excess continuous time poles is also discussed.

The implementation aspects of the self-tuning regulators have been discussed in [13]. The difference between the simulation algorithms and the practical algorithms as well as some aspects of implementation have also been analysed in this reference.

Recently there have been a flurry of activities in applying neural networks to control systems. An attempt has been made to utilize neural networks as implicit self tuning controllers in [14]. While the approach proposed by the authors could also be used to control non-linear systems, examples of only the linear systems have however been provided. But the minimum variance strategy fails when control of non-minimum phase systems is considered. Also, disturbances are incorporated by using an extra neural network which added extra computations.

Some methods of using the neural networks as explicit self tuning controllers have been given in [15] where nonlinear systems are treated and means of utilizing

multilayer and recurrent networks are given. Only minimum phase plants of relative degree unity have been considered. But the explicit methods of self tuning are computationally at a disadvantage when compared to the implicit ones. This is because of the indirect way of the controller synthesis. Further, the structure of the plant is required to be known.

Utilization of neural networks for self-tuning of nonlinear systems has also been addressed in [16] though the author has only considered a plant with a special structure. This approach however may unnecessarily add to the computations in many cases.

An excellent survey of the importance of neural networks from a control systems perspective and also the future areas for research have been given in [17]. The main focus is on the promise of artificial neural networks in the realm of modelling, identification and control of nonlinear systems.

The design of practically viable controllers using neural networks, based on results in nonlinear control theory has been dealt with in [18]. In [19], the objective has been to clarify the neural network characteristics by comparison with adaptive control theory. Stability analysis of the neural network controller has also been touched upon.

The convergence properties of the neural networks have been addressed in [20]. A new algorithm which has better convergence properties than the classical backpropagation algorithm has also been given. The increase in computational burden using the given algorithm is also quite minimal.

Improved backpropagation algorithms for faster convergence have also been given in [21]. It is shown that the proposed algorithms can avoid convergence to a local

minimum in most cases. Simulation results are also provided to support the claims in [21]. The proposed algorithms are also simple and in many cases the computational storage requirements and computational load are quite comparable to those required by the classical backpropagation algorithm.

The issue of utilizing neural networks in control systems continues to be a hot topic of research these days. This led for example, to complete issues of the IEEE control systems magazine being devoted to it in the last few years. Quite a few of the papers in these issues deal with the adaptive control of systems using neural networks [16], [22]. [23].

Chapter 3

NEURAL NETWORKS AND SELF TUNING CONTROLLERS

SUMMARY: The fundamentals of neural networks are described. Architecture and training of multilayered neural networks utilizing the backpropagation algorithm are included. In the later part of this chapter the rudimentary principles of the self-tuning controller are also presented.

Neural Networks can be described as an attempt by humans to create an artificial brain. In the current stage of development of Neural networks, though, it would be more apt to describe them as a human attempt to mimic the way the brain is supposed to do things. This is in order to harness its versatility and its ability to infer knowledge from incomplete or distorted information.

Learning about neural networks requires a new vocabulary. A neural network is not programmed, it's "taught". A neural network's speed is measured not in terms of instructions per second but in terms of interconnections between neurons per second.

The main motivation behind the development of the artificial neural network is the brain. The many different network paradigms and algorithms have been developed by researchers in such a way as to be analogous to the supposed functioning of the brain. But then the current knowledge about the brain's overall operation is very limited. This constrains the network designers to go beyond current biological knowledge, seeking structures that perform useful functions. In many cases, this necessary shift discards biological plausibility, the brain becomes a metaphor. Networks are produced that are biologically infeasible or require a highly improbable set of assumptions about brain anatomy and functioning.

Yet artificial neural networks evoke comparisons with the brain since their functions are often reminiscent of human cognition. Hence it remains profitable to understand something of the mammalian nervous system - an entity which successfully performs the tasks to which our artificial systems only aspire.

3.1 Biological Neuron

The human nervous system, built of cells called neurons, is of staggering complexity. An estimated 10^{11} neurons participate in perhaps 10^{15} interconnections over transmission paths that may extend to a meter or more. Each neuron is an electrically active cell. They interact with one another through the flow of local ionic currents between them. These ionic currents are driven by a voltage difference across the neuron's cell membrane.

A nerve impulse consists of a rapid voltage change which occurs in a localized section of a neuron's membrane. Once initiated this information propagates to adjacent areas along the length of a nerve fibre which links cell bodies, called somas,

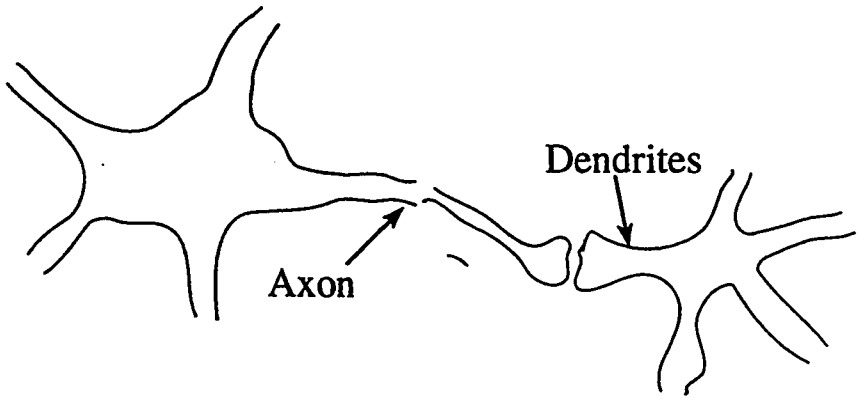


Figure 3.1: Biological Neuron

to networks. At the end of each output fibre (axon) the electrical nerve impulses are converted into chemical signals which cross a synaptic gap and which may initiate electrical impulses in the input fibres (dendrites) of the sink cells.

The resulting nerve impulses then travel along their dendrites to arrive at the target cells. Each cell body continually integrates the currents that arrive via its dendrites. Two types of integration effects that deserve mention are the *temporal summation* and the *spatial summation* as shown in fig. 3.2. Temporal summation is a mechanism which depends on the pulse frequency of an incoming pulse train over a single dendrite. Spatial summation is an interaction that refers to the roughly simultaneous arrival of pulses from a number of distinct inputs [24]. Each input travels over a different dendrite.

This seemingly baroque mechanism is surprisingly powerful since not only is each neuron an independent simple processor but also at any instant of time a tremendous number of neurons can be active thus making the network mechanism highly parallel. The functioning of the artificial neural network closely parallels the

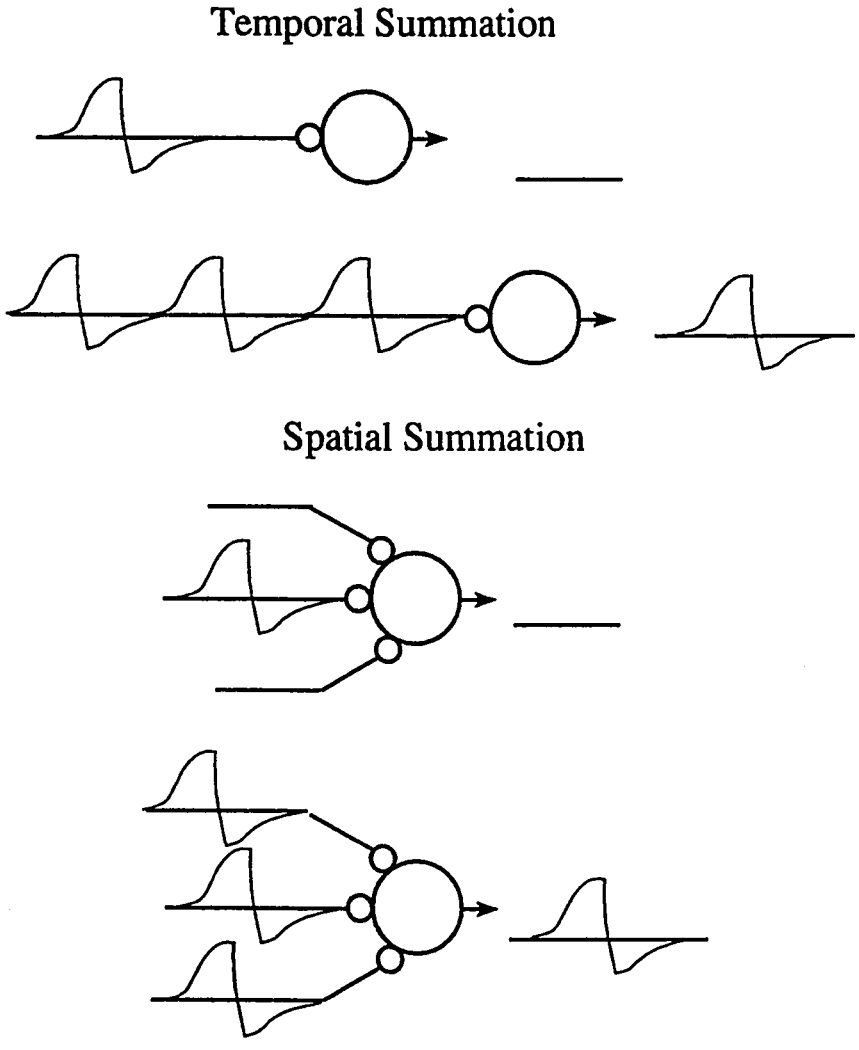


Figure 3.2: Cellular integration

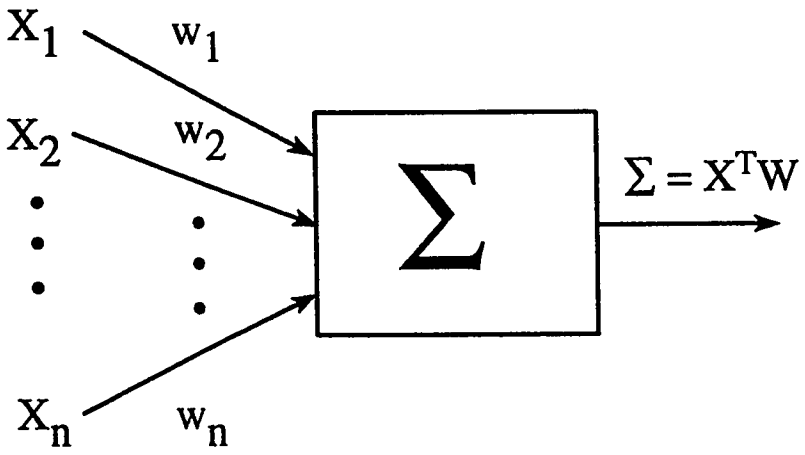


Figure 3.3: Artificial Neuron

mechanism outlined above.

3.2 Artificial Neuron

The artificial neuron was designed to mimic the characteristics of the biological neuron as understood by the human mind. In essence a set of inputs are applied to each neuron. Each input is applied over a separate link as shown in fig. 3.3. A weight is associated with each link. Each input is multiplied by the weight of the corresponding link through which it is applied, analogous to a synaptic strength. All the weighted inputs are then algebraically summed to determine the activation level of the neuron. A model that implements this idea is shown in fig.3.3. The different inputs X_1, X_2,..., X_n are applied to one link each. These are multiplied by the link weight and finally algebraically summed up.

Despite the diversity of network paradigms, nearly all are based on this config-

uration. A set of inputs labeled X_1, X_2,..., X_n is applied to the artificial neuron. Each signal in then multiplied by an associated weight w_1, w_2,..., w_n. Finally it is applied to the summation block. The summation block algebraically adds up all the inputs and sends the resultant output which is denoted as Σ. In vector notation if $\mathbf{X}$ denotes a vector whose i^{th} component is X_i and W is a weight vector whose i^{th} component is w_i, then, the resultant net can be expressed as

$$\Sigma = X^T W \tag{3.1}$$

3.2.1 Activation function

The signal Σ is usually further processed by an activation function f as shown in fig.3.4 to produce the neuron's output signal Y. Generally the activation function used is the so-called "squashing function" which, in the case of a sigmoid, is expressed mathematically as

$$Y = F(\Sigma) \tag{3.2}$$

$$\text{where} \quad F(\Sigma) = \frac{1}{1+e^{-\Sigma}} \tag{3.3}$$

Fig.3.5 depicts the sigmoidal logistic function. Another nonlinear activation function which is commonly used is the hyperbolic tangent. It is shown in fig.3.6. Mathematically it is expressed as

$$\begin{aligned} F(\Sigma) &= tanh(\Sigma) \\ &= \frac{1-e^{-\Sigma}}{1+e^{-\Sigma}} \end{aligned} \tag{3.4}$$

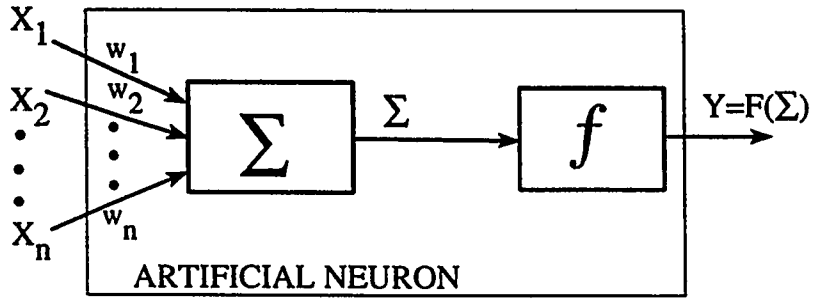


Figure 3.4: Artificial Neuron with Activation Function

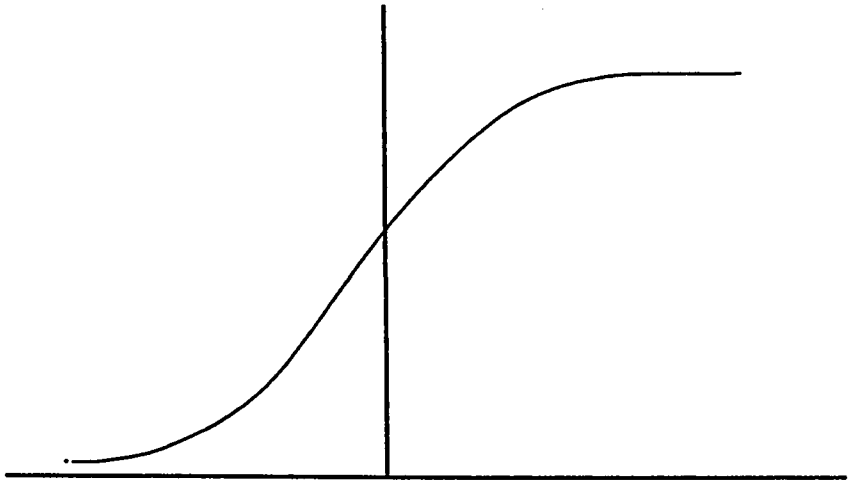


Figure 3.5: Sigmoidal logistic function

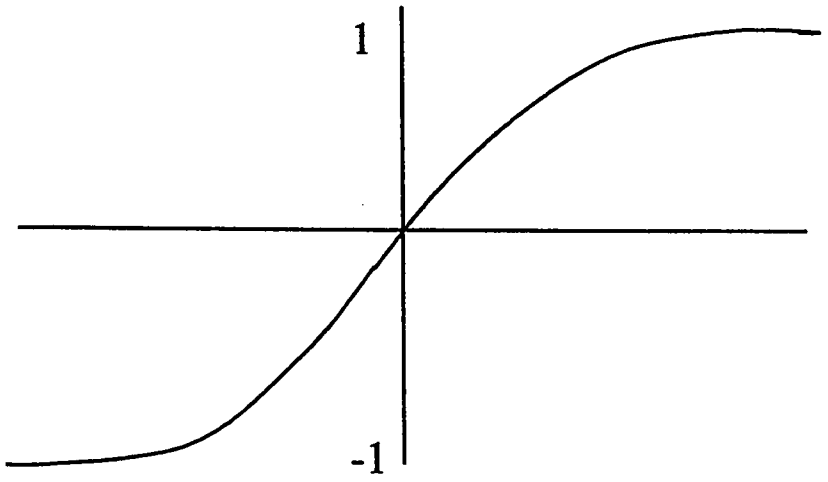


Figure 3.6: Hyperbolic tangent function

3.2.2 Single layered network

The power of neural computation comes from connecting the neurons into networks. The simplest network consists of a group of neurons arranged in a layer as shown in fig.3.7. Neurons present in the same layer are not connected with each other. The circular nodes on the extreme left in fig. 3.7 serve only to distribute the input. They perform no computation and hence are also referred to as the zeroth layer. The set of inputs $\mathbf{X}$ has each of its elements connected to each neuron through a link, each of which has a weight associated with it. Each neuron simply outputs the algebraic weighted sum of the inputs to the network, acted upon by the activation function. Actual networks may also have many of the connections deleted.

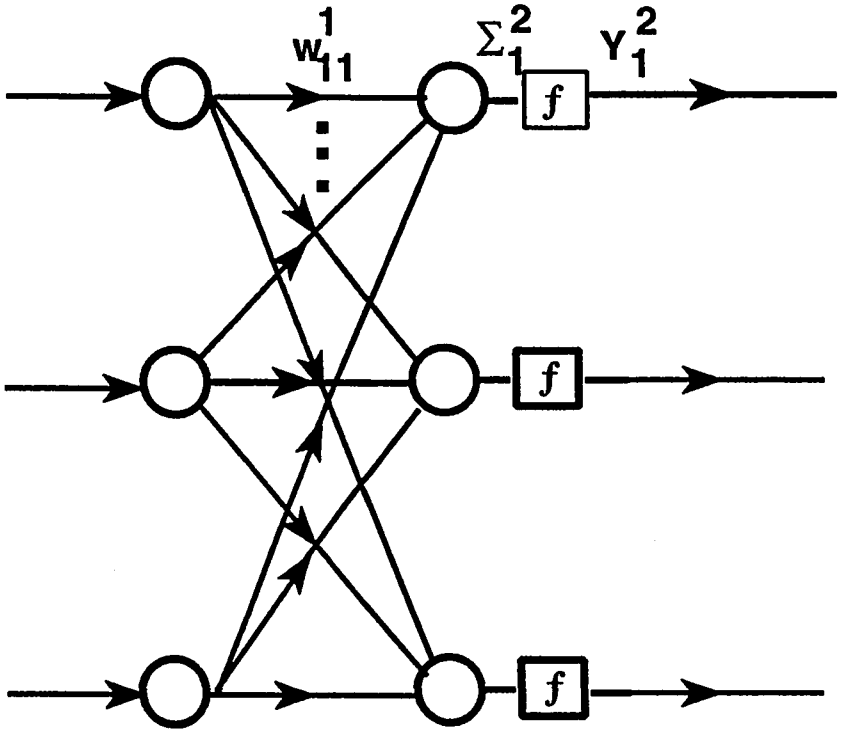


Figure 3.7: Single-Layer Neural Network

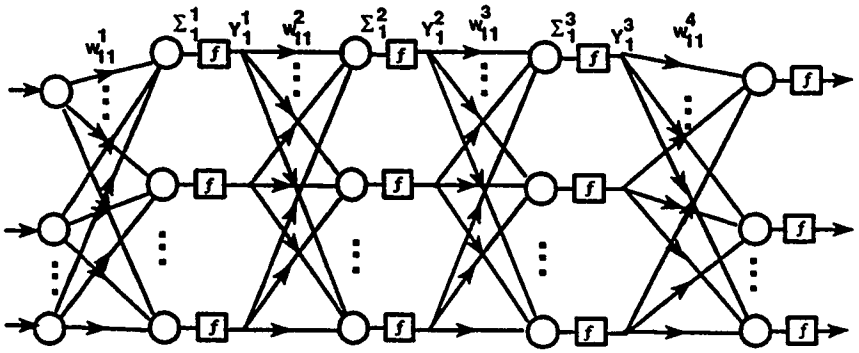


Figure 3.8: Multilayered Neural Network

3.2.3 Multilayer Artificial Neural Networks

Greater computational capabilities are offered by large, more complex so-called multilayered networks as shown in fig.3.8. Each neuron in a layer is completely connected with all the neurons in the next layer. Neurons in the same layer are not connected. This is the feedforward neural network architecture. Although networks have been constructed in every imaginable configuration, arranging neurons in layers mimics the layered structure of certain portions of the brain. Multilayered neural networks have been proven to have capabilities beyond those of a single layer. But the nonlinear activation functions are vital to the expansion of the network's capabilities beyond that of the single-layer network since without these functions multilayered networks provide no advantage in flexibility over a single layer network [25].

Much of the current fascination with neural networks has to do with their ability to learn. The most popular learning algorithm today is the backpropagation, which can be implemented rather easily on a microcomputer [26]. The network is trained so that application of a set of inputs produces a desired or at a least consistent set of outputs. Each such input set is referred to as a vector.

3.3 Backpropagation training algorithm

The proposition of the backpropagation algorithm has played a central role in the resurgence of interest in artificial neural network. This is mainly because of the limited capability of single layered neural network. Backpropagation is a systematic method for training multilayered artificial neural network.

The backpropagation algorithm is quite simple to implement. The backpropagation algorithm is a gradient descent search technique. Let Y_d be the target output of the network and Y be the actual output of the network. The difference between the target output and the actual output is the error denoted as e_k. Define a cost function as

$$\begin{aligned} J &= \frac{1}{2}||e_k||^2 \\ &= \frac{1}{2}||Y_d - Y||^2 \end{aligned} \tag{3.5}$$

The aim of the backpropagation algorithm is to minimize the cost function J through an iterative search in the weight space W. This is because the actual output Y is a function of its weights.

The well known delta rule for updating the weights is a first order gradient descent algorithm moving in the negative gradient direction. Using the delta rule we have

$$W_{k+1} = W_k - \eta \left[\frac{\partial^+ J}{\partial W}|_{W=W_k}\right]^T \tag{3.6}$$

where η is a small positive quantity chosen arbitrarily and $\partial^+ J/\partial W$ denotes the ordered derivative [27]. In case of partial derivative all terms other than the variable of interest are assumed constant. This would not be of much use especially in case

of ordered system of equations. The neural network is also an ordered system and hence the concept of partial derivative is not mathematically correct. Thus arises the concept of ordered derivative.

Differentiation of 3.5 gives

$$\frac{\partial^+ J}{\partial W} = -\frac{\partial^+ Y}{\partial W}.e_k \tag{3.7}$$

$$= -\frac{\partial^+ Y}{\partial \Sigma}.\frac{\partial \Sigma}{\partial W}.e_k \tag{3.8}$$

The activation function mostly used in the application of neural networks in control systems is the hyperbolic tangent given by 3.4. It also has a simple derivative, a fact that is used in the implementation of the backpropagation algorithm. In the case of the single layer network, by differentiating 3.4 one gets.

$$\frac{\partial^+ Y}{\partial \Sigma} = \frac{1}{2}[1+Y][1-Y] \tag{3.9}$$

Further from equation 3.1 we have

$$\frac{\partial^+ \Sigma}{\partial W} = X \tag{3.10}$$

Hence substituting 3.8, 3.9 and 3.10 in 3.6, one gets

$$W_{k+1} = W_k + \eta.\frac{1}{2}[1+Y][1-Y].X.e_k \tag{3.11}$$

Backpropagation algorithm can be applied to networks with any number of layers, however only two layers of weights are necessary to demonstrate the algorithm. The backpropagation algorithm suffers though from the disadvantage that no systematic guideline is available for the choice of η. Although with a conservative choice of η, convergence to a local minimum is guaranteed, the convergence rate is, most of the time, painfully slow.

3.3.1 Training Overview

The objective of training the network is to adjust the weights so that application of a set of inputs produces the desired set of outputs. Training assumes that each input vector is paired with a target vector representing the desired output. Together these are called a training pair. Usually a network is trained over a number of training pairs. Before starting the training process, all of the weights must be initialized to small random numbers. Training the backpropagation network requires the steps that follow.

1. Select the next training pair from the training set; apply the input vector to the network input terminals.

2. Calculate the output of the network.

3. Calculate the difference between the network output and the desired output. This is the error.

4. Adjust the weights of the network in a way that minimizes the error.

5. Repeat steps 1 through 4 for each vector in the training set until the error for the entire set is acceptably low.

Steps 1 and 2 constitute a forward pass while steps 3 and 4 comprise the reverse pass.

Forward Pass

Let matrix W denote the weights between neurons. Let $\mathbf{X}$ be an input vector to a given layer of neurons whose i^{th} component is X_i. Then the output of the given

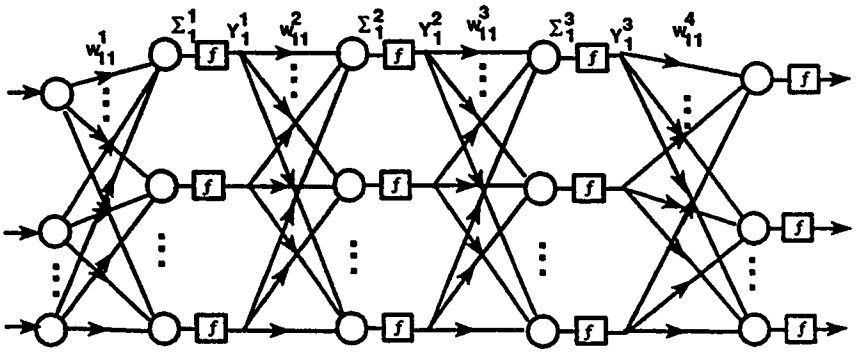


Figure 3.9: Forward Pass

layer of neurons is given by the the vector Y where

$$Y = F(X^t W) \tag{3.12}$$

and F(.) is the hyperbolic tangent function as given by equation 3.4. The output vector of one layer is the input vector to the next. Hence in order to calculate the outputs of the final layer the above equation is applied repeatedly to each layer between the network's input and its output.

Reverse Pass

Given an input-output pair the error which is used to adjust the weights is the difference between the actual and desired outputs of the network. The error is then multiplied by the derivative of the squashing function for that layer's neuron k thereby producing the backpropagated error for the k^{th} neuron in the i^{th} layer, Ω^i_k, where

$$\Sigma_k = (X^T_i W)_k \tag{3.13}$$

$$\Omega^i_k = F^{-1} e_k = \frac{1}{2}[1 - Y_k][1 + Y_k] e_k \tag{3.14}$$

$$e_k = Y_d - Y_k \tag{3.15}$$

The above method of computing the backpropagated error is done only for the neurons in the outermost i.e the output layer. Then the backpropagated error is multiplied by the output of the neuron j, the source neuron for the weight in question. This product is in turn multiplied by a training rate coefficient η and the result is added to the weight. An identical process is performed for each weight proceeding from a neuron in the hidden layer to a neuron in the output layer.

Mathematically

$$\Delta w_{pq}^k = \eta \Omega_q^k O_p^j \tag{3.16}$$

$$w_{pq}^k(n+1) = w_{pq}^k(n) + \Delta w_{pq}^k \tag{3.17}$$

where

$w_{pq}^k(n)$ = the value of a weight from neuron p in the hidden layer to neuron q in the output layer at step n.

Y_p^j = output of neuron p in the hidden layer j.

In order to adjust the weights feeding into the hidden layer, the values of the backpropagated error associated with each of the nodes in the output layer are first propagated from the output layer to the input layer. Each of the weights connecting the output layer to the hidden layer are multiplied by the backpropagated error value of the neuron to which it connects in the output layer. The value of the backpropagated error needed for the hidden-layer neuron is produced by summing all such products and multiplying by the derivative of the squashing function of the corresponding hidden layer neuron. Thus

$$\Omega_p^j = 1/2(1 - Y_p^j)(1 + Y_p^j)(\sum_q \Omega_q^k w_{pq}^k) \tag{3.18}$$

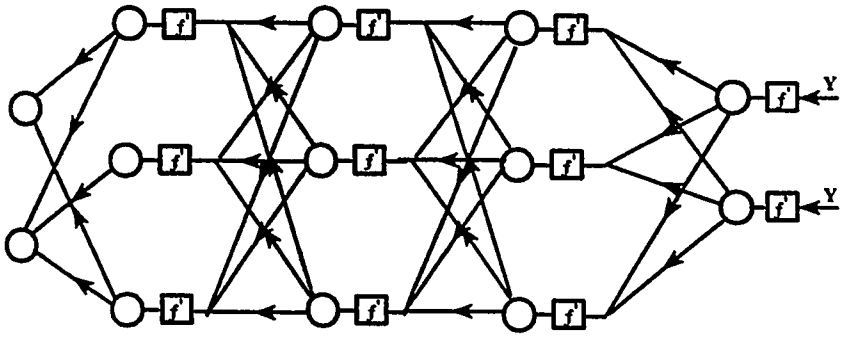


Figure 3.10: Reverse Pass

Using the Ω_p^j (the value of the backpropagated error for the p^{th} neuron in the j^{th} layer) the weights feeding the first hidden layer can be adjusted using equations 3.16 and 3.17 modifying the indices to indicate the correct layers. This is shown in fig.3.10.

In order to speed up the convergence rate of the backpropagation algorithm we have used the following approaches.

3.4 Improved backpropagation algorithm

3.4.1 Variable scalar learning rate

This algorithm has been developed by Ahmed in [21]. It is based on error linearization in the weight space.

The backpropagation algorithm determines a set of weights W that minimizes the sum squared error for the training patterns. But since this cannot be done analytically, a numerical optimization procedure is performed utilizing various approximations.

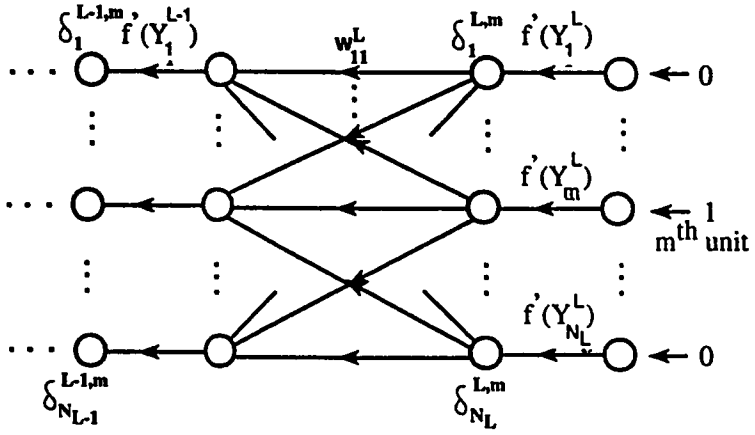


Figure 3.11: Principle of Convergence acceleration in the backpropagation algorithm

We assume that there exists at least one value W^o, the desired W such that

$$Y(W^o, X) = Y_d(X).$$

Assuming that W_k lies in the vicinity of W^o and using a first order approximation of W_k about W^o we have

$$Y_d(x) = Y(W^o.X) \approx Y(W_k, X) + \frac{\partial^+ Y}{\partial W}|_{W=W_k}.(W^o - W_k) \quad (3.19)$$

$$\approx Y(W_k, X) + \Gamma_k^T.\tilde{W}_k \quad (3.20)$$

where

$$\Gamma_k^T = \frac{\partial^+ Y}{\partial W}|_{W=W_k}$$

and

$$\tilde{W}_k = (W^o - W_k)$$

Using equation 3.5 and 3.20 the error in each iteration can be written as

$$e_k = Y_d(X) - Y(W_k, X) \quad (3.21)$$

$$= \Gamma_k^T . \tilde{W}_k \tag{3.22}$$

Combining 3.6, 3.7 and 3.22 we have

$$W_{k+1} = W_k + \eta_k \Gamma_k \Gamma_k^T \tilde{W}_k \tag{3.23}$$

It has been shown in [21] that the following choice

$$\eta_k = \frac{\mu}{\gamma} \quad , 0 \leq \mu \leq 2 \tag{3.24}$$

$$\gamma = tr(\Gamma_k . \Gamma_k^T) = ||\Gamma_k||^2 \tag{3.25}$$

ensures that

$$||\widetilde{W}_{k+1}||^2 \leq ||\widetilde{W}_k||^2 \tag{3.26}$$

as desired.

In order to avoid numerical problems as $\Gamma \rightarrow 0$, equation 3.24 is modified as

$$\eta_k = \frac{\mu}{\gamma + \epsilon} \tag{3.27}$$

where ϵ is a small quantity which in our simulation is taken as 0.001.

The optimum value of μ is 1 but a more conservative expression should be used at the initial stages where the linearization approximation is poor. Hence the expression proposed is

$$\mu(k+1) = 1 - \mu_r[1 - \mu(k)] \tag{3.28}$$

where $\mu(k)$ is the value of μ at the k^{th} iteration. μ_r is the rate of transition from $\mu(0)$ to 1 and $\mu(0)$ is taken much smaller than 1.

The above expression for η has been developed under the assumption of there being no disturbance in the system. Hence in this case the optimum value of the cost function J^* is 0.

But in the presence of noise, the error e(k) will include both the noise (i.e. the disturbance) and the deviation due to the final suboptimal weights. When the weights approach their optimal values e(k) will approach the noise terms. Hence, the optimum value of the cost function J^* will be a nonzero positive quantity. Then a linearized approximation of e(k) in the vicinity of W^o, where W^o is the optimal value of the weights, is

$$e_k = e_k^* + \Gamma_k^T \tilde{W}_k \tag{3.29}$$

$$\tag{3.30}$$

where e_k^* represents the noise term. In this case 3.23 is modified to

$$W_{k+1} = W_k + \eta_k \Gamma_k \Gamma_k^T \tilde{W}_k + \eta_k \Gamma_k e_k^* \tag{3.31}$$

Subtracting both sides from W^o we have

$$\tilde{W}_{k+1} = \tilde{W}_k - \eta_k \Gamma_k \Gamma_k^T \tilde{W}_k - \eta_k \Gamma_k e_k^* \tag{3.32}$$

Taking the Euclidean norm we have

$$\begin{aligned} ||\tilde{W}_{k+1}||^2 = ||\tilde{W}_k||^2 - 2[\eta_k \xi_k \tilde{W}(k)]^T \tilde{W}(k) + \eta_k^2 \tilde{W}^T(k) \xi_k^2 \tilde{W}(k) \\ -[2\tilde{W}^T(k)(I + \eta_k \xi_k)\eta_k \Gamma_k e_k^*] + \eta_k^2 \gamma_k' e_k^{*T} . e_k^* \end{aligned} \tag{3.33}$$

where

$$\xi_k = \Gamma_k \Gamma_k^T \tag{3.34}$$

$$\gamma_k' = \Gamma_k^T \Gamma_k = tr[\xi_k] \tag{3.35}$$

$$\sigma_k^2 = E(e_k^{*T} . e_k^*) \tag{3.36}$$

By taking expectations of both sides of equation 3.33 and assuming the noise sequence to be zero mean and uncorrelated with the network inputs, the fourth term on the right hand side of equation 3.33 becomes zero. This gives

$$E[||\tilde{W}_{k+1}||^2] = E\{||\tilde{W}_k||^2 - 2\eta_k||\tilde{W}_k||^2_{\xi_k} + \eta_k^2\gamma_k'[\sigma_k^2 + ||\tilde{W}_k||^2_{\xi_k}]\} \quad (3.37)$$

$$(3.38)$$

where

$$||\tilde{W}_k||^2_{\xi_k} = \tilde{W}_k^T \xi_k \tilde{W}_k \quad (3.39)$$

Defining

$$r_k \equiv \frac{\tilde{W}_k}{\sigma}$$

one gets

$$E[||r_{k+1}||^2] = ||r_k||^2 - 2\eta_k||r_k||^2_{\xi} + \eta_k^2\gamma'[1 + ||r_k||^2_{\gamma}] \quad (3.40)$$

Let us make the approximation that

$$||r_k||^2_{\xi} \approx [\lambda_{av}(\xi)][||r_k||^2] \quad (3.41)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\lambda_i||r_k||^2 \quad (3.42)$$

$\lambda_{av}(\xi)$ is the average of the eigenvalues of ξ. Also

$$\sum_{i=1}^{n}\lambda_i = \text{Trace}[\xi] = \gamma' \quad (3.43)$$

Hence

$$||r_k||^2_{\xi} = \frac{\gamma'}{n}||r_k||^2 \quad (3.44)$$

where n is the total number of system parameters. Hence using this we have

$$E[||r_{k+1}||^2] = E\{||r_k||^2 - 2\eta_k \frac{\gamma'}{n}||r_k||^2 + \eta_k^2 \gamma' [1 + \frac{\gamma'}{n}||r_k||^2]\} \quad (3.45)$$

To ensure $E[||r_{k+1}||^2] \leq E[||r_k||^2]$ we require

$$\eta_k \geq 0$$

and

$$2\eta_k \frac{\gamma'}{n}||r_k||^2 \geq \eta_k^2 \gamma' [1 + \frac{\gamma'}{n}||r_k||^2] \quad (3.46)$$

$$\eta_k \leq \frac{2||r_k||^2/n}{1 + \gamma'||r_k||^2/n} \quad (3.47)$$

Letting s(k) = $||r_k||^2/n$ we get

$$0 \leq \eta_k \leq \frac{2s(k)}{1 + \gamma' s(k)} \quad (3.48)$$

or

$$\eta_k = \frac{\mu s(k)}{1 + \gamma' s(k)} \quad (3.49)$$

where

$$0 \leq \mu \leq 2$$

A recursive expression for s(k) is also derived in [28] and is given by

$$s(k+1) = s(k)[1 - (2 - \mu)\gamma' \eta_k] \quad (3.50)$$

The relaxation factor μ can be advantageously used to modify the adaptation of the parameters. The equations 3.49 and 3.50 constitute the algorithm for the computation of the learning rate for the noisy case.

3.4.2 Variable matrix learning rate

The formulae for the matrix learning rate have their basis in the recursive prediction algorithm [29]. Using the matrix learning rate the weight upgradation scheme is given as

$$P(k+1) = \{P(k) - \frac{P(k)\Gamma\Gamma^T P(k)}{\lambda + \Gamma^T P(k)\Gamma}\}/\lambda \tag{3.51}$$

$$W(k+1) = W(k) + \frac{P(k)\Gamma}{\lambda + \Gamma^T P(k)\Gamma} e \tag{3.52}$$

P(0) may be taken as the identity matrix. Further λ can also be varied using the expression

$$\lambda(k) = \lambda(k-1) + \{\lambda_\infty - \lambda(k-1)\}\lambda_r \tag{3.53}$$

The initial value of λ is taken about 0.97, λ_∞ is taken as 1 and λ_r has a value of about 0.99.

3.5 Self Tuning Controllers

The fundamental components of a self-tuning system are discussed here. The concept of a self-tuning control system is basically very simple. It is based upon the certainty equivalence principle. The two essential components of any self-tuning control system are the identification technique and the control law. By combining any identification scheme with a particular control law, a wide spectrum of self tuning algorithms can be developed.

In the case of conventional control schemes, the controllers developed are such that their parameters do not change with time. In other words the parameters of these controllers are time-invariant. Thus an implicit assumption made in this case is that the parameters of the system to be controlled are constant. Hence it's assumed that a controller with constant parameters will accomplish the goals adequately. But this approach is not the best when dealing with actual systems. This is so because in everyday life, various factors such as changing environmental conditions, varying quality of the raw material as well as the effects of wear and tear on the plant performance cannot be neglected since these factors cause the plant parameters to drift and sometimes radically change. To take care of the resulting deterioration in the plant performance, one has to either change the controller which is a costly proposition or to resort to retuning the controller which is the approach normally used in industry.

In order to automatically overcome these practical problems associated with the control of an actual system, the self-tuning regulator has been proposed. With such a regulator, the unknown parameters of the system to be controlled are estimated recursively. This estimation can be done using any of the established parameter

estimation techniques. Once the plant parameters are estimated, then the controller parameters are computed using any control law. The operation of a self-tuning controller is schematically shown in fig.3.12.

There are basically two broad class of algorithms depending on the intricacy of the design calculations involved. They are

- Explicit Self-tuning Controllers
- Implicit Self-tuning controllers

Explicit Self-tuning Controllers

The simplest conceptual strategy is to parameterize the system in a natural way and calculate the controller parameters from the estimated plant parameters as shown in fig 3.13. This class of algorithms is commonly called **indirect or explicit.** This is because the evaluation of the control law is indirectly achieved via the system model. Thus an explicit process model is estimated in the first step. In the second step a design method is used to determine the parameters of the controller using the estimated parameters in the first step. These two steps are repeated at each sampling interval.

But the main disadvantage of this method is that the computations involved are many. Further it is also required that the structure of the system to be controlled be known.

Implicit Self-tuning Controllers

It is also possible to parameterize the system directly in terms of the controller parameters. Hence in this case, as soon as the parameters are identified the controller is directly obtained thereby obviating the need for computing the control law

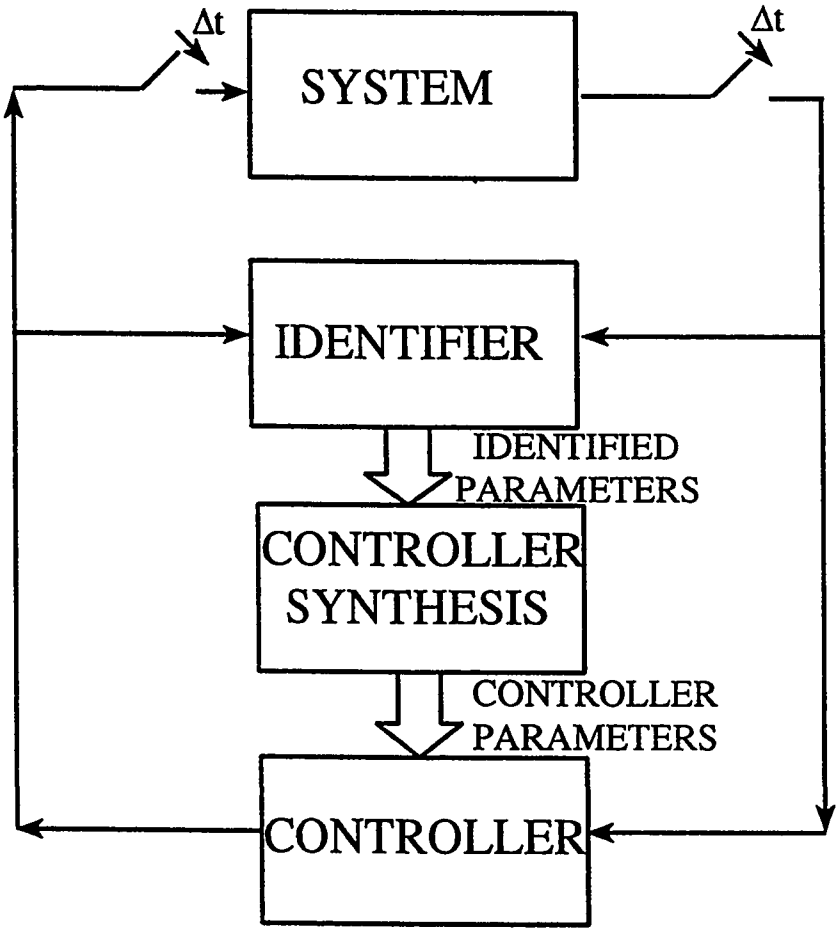


Figure 3.12: Self Tuning Controller

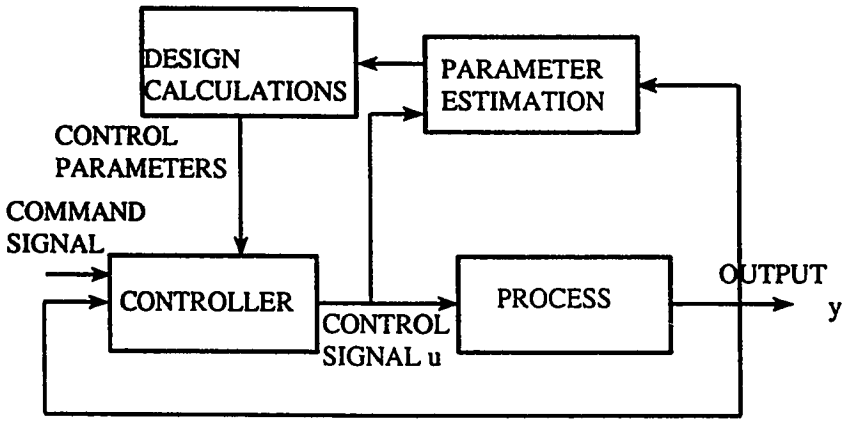


Figure 3.13: Explicit self tuning controller

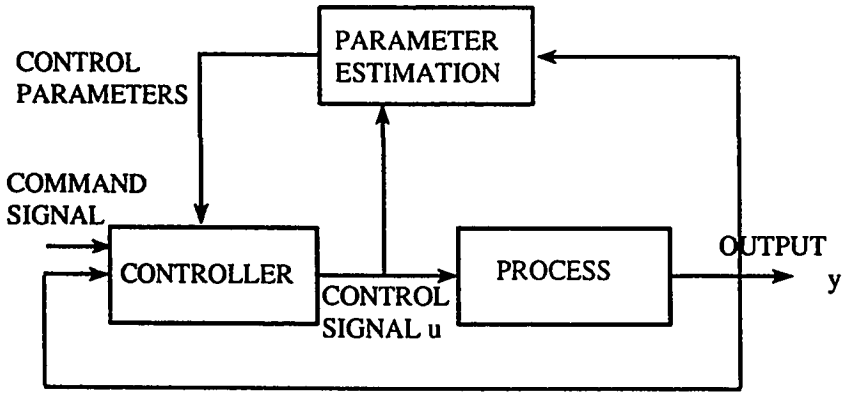


Figure 3.14: Implicit self tuning controller

again. Thus in this case the design calculations necessary to determine the controller parameters are simply the parameters in a one-step ahead predictor. This class of algorithms is commonly called **direct or implicit** because the control law is directly estimated as shown in fig.3.14.

One advantage of the implicit self tuning controllers(STC) over the explicit ones is that the design computations are eliminated from the use of the former ones since the controller parameters are directly estimated. But the number of parameters involved in the design of implicit STCs may be large compared with the explicit ones.

3.6 Parameter estimation

For the purpose of adaptive control of a system, recursive parameter identification has to be resorted to. This not only helps in tracking the changing system parameters but also updates the parameter estimates based upon the most recent data. Thus the parameter estimates are updated after receipt of each sample. The controller is then synthesized based upon the current estimate of the process. Moreover, recursive parameter identification methods require only a modest amount of information. Hence the demands on memory requirements are also less. Several techniques have been applied for estimating the parameters. Any of these could be used in the self tuning controller to be designed. Some of the recursive parameter estimation methods are

- Least squares
- Maximum likelihood

- Instrumental variables
- Stochastic approximation

Least Squares Method

This is a mathematical technique used to estimate the parameters of the plant. The experimental data are used to achieve a "best" fit model. The criterion used is the minimization of the sum-squared error. The error is defined as the difference between the actual observation and the estimated observation. The main advantages of this technique are that it is very simple to use and is not computationally demanding.

Maximum Likelihood Method

In the Maximum likelihood method a logarithmic likelihood function is maximized with respect to the parameter estimates θ. Under the assumption of gaussian distributed errors, computationally feasible algorithms can be obtained. However, for other distributions the algorithm may become quite complex and unsuitable for recursive estimation.

Instrumental variables Method (IV)

The ordinary least squares algorithm produces biased estimates in the presence of correlated noise. IV method gives a computationally simple procedure to eliminate the bias where the effect of noise is correlated out through the use of instrumental variables. The IV method has also been found useful in determining the system structures from noisy data.

Stochastic Approximation Method

The purpose of this estimation technique is to reduce the computational requirements of the recursive least squares method. This is done by minimizing the loss

function by a stochastic gradient method. It is identical to the recursive least squares algorithm except that the parameter gain adaptation is simpler.

The stochastic gain algorithms have computational advantages over the RLS ones since the computations involved are very straightforward. These methods are valuable in processing data which arrive is such large quantities that its entire storage may pose problems. Further it can also be shown that the least squares method is a special case of the stochastic approximation method. [28]

3.7 Control Strategies

The different control strategies which can be used in the design of a self-tuning controller are

- Minimum Variance technique
- Generalized Minimum Variance technique
- Pole assignment technique
- Predictive control

The first two techniques are derived from optimal control principles while the pole assignment technique is associated with classical control methods. Predictive control is also based on optimal control methods and can be said to be a more generalized version of the minimum variance technique.

Minimum Variance Technique

This was the technique originally used by Astrom and Wittenmark in [5]. The basic self tuning regulator has been designed in [5] for a situation where the control

problem could be characterized as a minimum-variance regulation problem or a set-point tracking one. The criterion in this case is the minimization of the variance of the output. The parameters of the plant are estimated recursively and the control law parameters are determined from the estimated model of the plant assuming the estimated model to be in close conformity with the actual model. This strategy of using the estimates as if they were the true parameters for the purpose of the design of the controller is called the certainty equivalence principle.

The deficiencies of this method are that it cannot be used to control plants which are nonminimum phase and also large and rapidly varying control action is demanded which may lead to the actuator saturation. Further if the number of integer time delays k is unknown or is variable then the minimum variance controller may experience problems if the model contains fewer delays than actually exist. In effect, the regulator then tries to introduce a pure time advance into the feedback. This causes extremely high feedback gains and may also result in instability [6].

Generalized Minimum Variance Technique

In this method the variance of a cost function is minimized. The cost function incorporates the system input, output and set-point. This technique replaces the system output by a generalized system output (or pseudo output). This approach was originally given by Clarke and Gawthrop in [10] and [11].

The advantages of this approach over the previous one is that it is applicable even for non-minimum phase plants. Also the control input required is smoother and smaller in magnitude. One of the reasons for this is that the input is also penalized and hence the control law in order to minimize the pseudo output demands an input which is much less in magnitude compared to the earlier method of minimum

variance.

Pole Assignment

The two earlier techniques of determining the control law are associated with the implicit or the direct self tuning method. The pole assignment technique is closely connected with the explicit or the indirect self-tuning control.

In this method the pole positions of the closed loop system are stipulated beforehand and once the system parameters are estimated the controller parameters are computed so as to achieve the desired closed loop performance. The position of the poles is decided upon based on various factors such as the transient performance desired from the system in conjunction with the controller, the system limitations etc.

While this method is suitable even for non-minimum phase plants, it is however disadvantageous since it leads to extra computations- a fact which is critical to the practical implementation of the controller to be designed.

Predictive Control

The minimum variance and the generalized minimum variance techniques are quite sensitive to the correct determination of the model order and delay. Also the pole placement method may fail if the estimated model polynomials aren't relatively coprime [30].

Hence long range predictive control methods were proposed. These methods deal with the behavior of the system over a horizon in the future as opposed to the minimum variance where the system behavior at a single point in the future is of interest.

Several control scenarios were proposed for long range prediction. Depending on the future control, forecasts of the process output over a long range horizon are made. Then that control scenario is selected which brings the predicted process output back to the setpoint in a predefined "best" way. The selected control strategy is applied as the real process input at the current moment. The whole procedure is repeated again at the next sampling instant [31].

Chapter 4

SELF TUNING CONTROL USING NEURAL NETWORKS

SUMMARY: Motivation for the use of neural networks in control is provided. The schemes adopted for the self tuning control of inverse stable as well as inverse unstable plants are also explained. Implementation diagrams of the proposed schemes are also provided.

Although nonlinear system theory has advanced considerably in recent years, applications have been mostly heuristic in nature. Mathematical systems theory as developed over the last few decades, has been mainly developed for linear systems. In order to study nonlinear systems many different models have been proposed such as the Wiener model, Hammerstein model, Volterra series, NARMAX models etc. The NARMAX model has been shown to be the most general one for the study of a wide class of discrete-time non-linear systems [32]. Mathematical functions can only approximately describe a real world system and even as such, can be very complex. Hence in practice a chosen model set of known functions is used for the modelling of

a real world system. But then the model set must be capable of describing the real world system to within an arbitrary accuracy. Mathematically this requires that the set be dense in the space of continuous functions.

The famous approximation theorem of Weirstrass [33] states that any function in the space of continuous real-valued functions defined on the interval [a,b] denoted as C[a,b] can be approximated arbitrarily closely by a polynomial. The set of polynomials as a result, is dense in C[a,b].

In recent years a lot of research activity has centered around neural networks. These networks are being applied to many areas of engineering and science. Two classes of neural networks which hold much interest from a system theoretic point of view are the multilayer neural networks and the recurrent networks. Neural networks which have a remarkable learning ability can be viewed as a class of functional representations. As a result they can be considered to be dense in the space of continuous functions and hence can be used for the modelling of nonlinear systems. In fact it has been shown by Cybenko [34] and Hornik et al. [35] that any continuous mapping over a compact domain can be approximated as accurately as necessary by a feedforward neural network, even with only one hidden layer. This implies that given any $\epsilon > 0$ a neural network NN with a sufficiently large number of nodes can be determined such that

$$||f(x) - NN(x)|| < \epsilon \quad for \ \ all \ \ x \ \in D$$

where f is the function to be approximated, D is a compact domain of a finite dimensional normed vector space and ||.|| denotes any suitable norm.[18]. This provides the necessary theoretical basis for modelling nonlinear systems by neural networks.

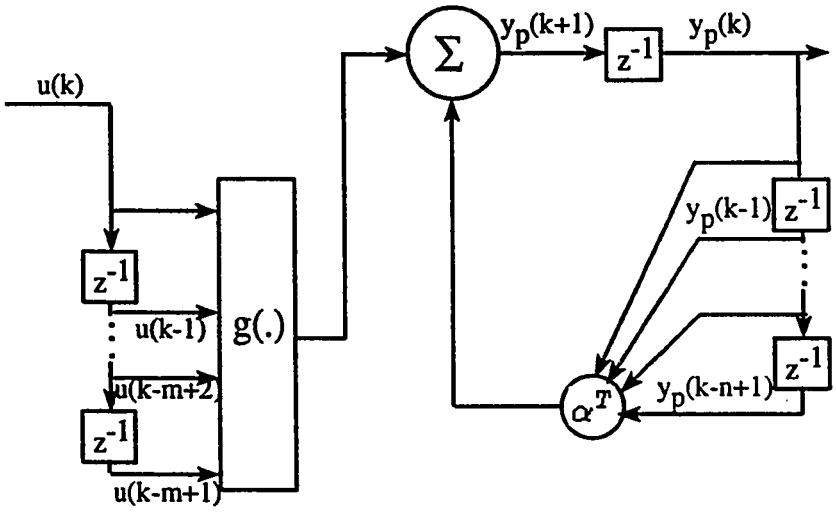


Figure 4.1: Class I

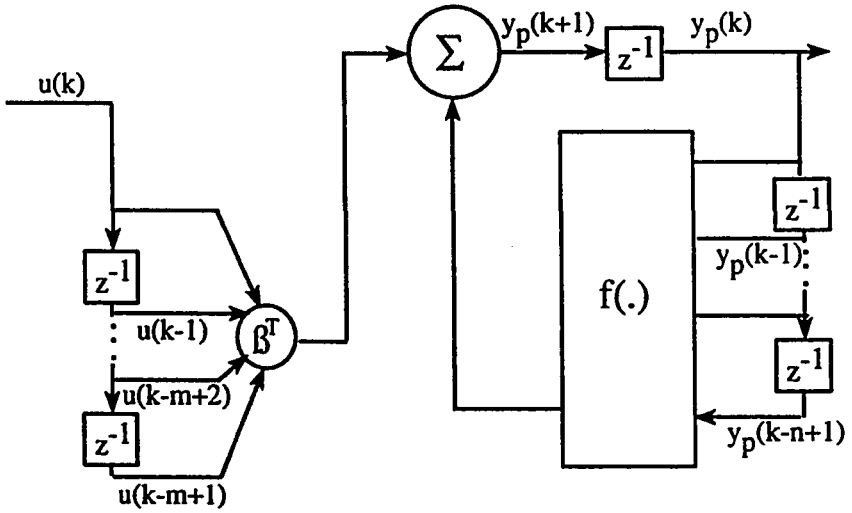


Figure 4.2: Class II

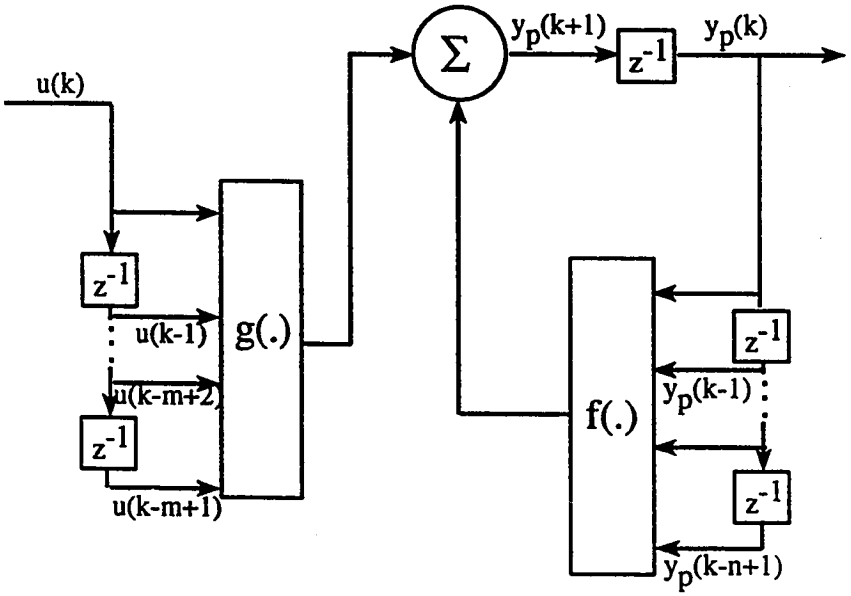


Figure 4.3: Class III

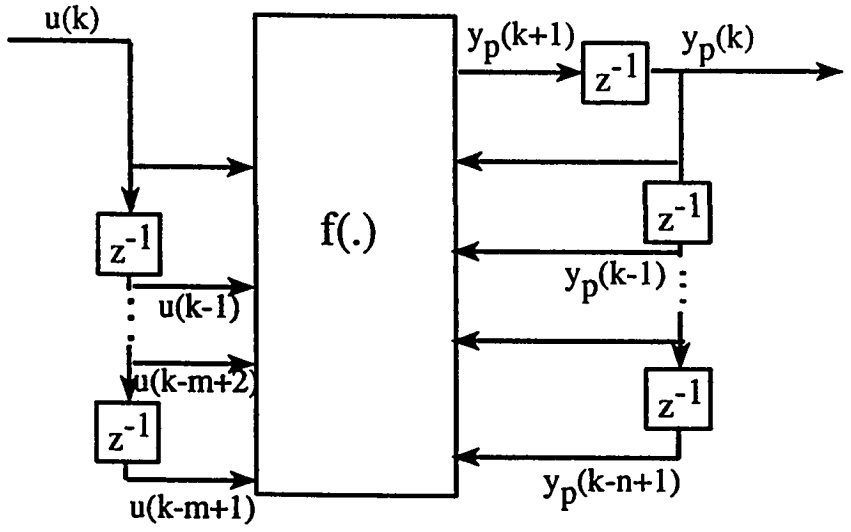


Figure 4.4: Class IV

Another factor responsible for the widespread use of feedforward neural network is the simplicity of the generalized delta rule (the backpropagation algorithm). The above two factors are also responsible for making neural networks attractive as system identifiers and controllers.

4.1 Identification and Control

Nonlinear SISO plants have been represented using many representations. They may be categorized into four different classes [15] which can be described as follows.

CLASS I

$$y_p(k+1) = \sum_{i=0}^{n-1} \alpha_i y_p(k-i) + g[u(k), u(k-1), \ldots, u(k-m+1)] \tag{4.1}$$

CLASS II

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \cdots, y_p(k-n+1)\right] + \sum_{i=0}^{m-1} \beta_i u(k-i) \tag{4.2}$$

CLASS III

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \cdots, y_p(k-n+1)\right] + g\left[u(k), \cdots, u(k-m+1)\right] \tag{4.3}$$

CLASS IV

$$y_p(k+1) = f\left[y_p(k), y_p(k-1), \cdots, y_p(k-n+1); u(k), \cdots, u(k-m+1)\right] \tag{4.4}$$

where $[y_p(k), u(k)]$ represent the output-input pair of the SISO plant at time k, and g:$R^m \rightarrow R$ in classes I and III, f:$R^n \rightarrow R$ in classes II and III and f:$R^{n+m} \rightarrow R$ in class IV are differentiable functions of their arguments. As for α and β, these are the parameters indicating the linear dependency of the output on the various delayed terms.

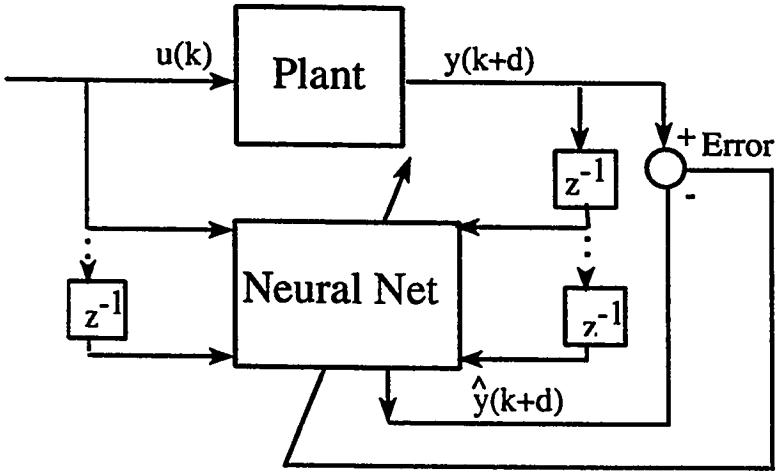


Figure 4.5: Series-parallel model

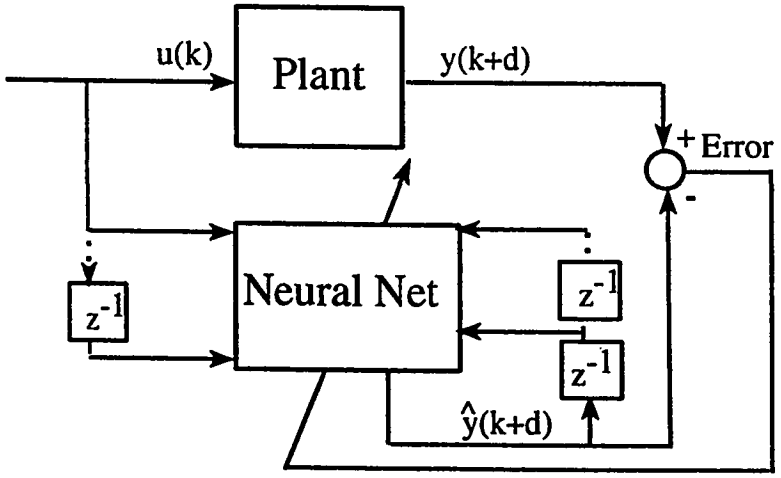


Figure 4.6: Parallel model

One aspect of dealing with dynamic or time series data using neural networks is that both present and past data have to be used to train the neural network. As a result, two different learning schemes can be used to model dynamic systems. In fact these schemes have been motivated by models used in the adaptive systems literature dealing with linear systems.

One possibility is to use the feedforward neural networks and use the actual past outputs of the system as the inputs to the network. This is referred to as the series-parallel or equation error model in the literature.

Another possibility is to use the estimated past outputs from the neural network as the current inputs to the neural network. This is referred to as the parallel or output error model. The two configurations are shown schematically in the figures 4.5 and 4.6.

There have been some problems associated with the parallel model [15]. In fact from the algorithm stability point of view the series-parallel model has been found to be preferable. The parallel scheme also has been found to have less flexibility. A stability check is also needed to be done continuously which may make the algorithm stagnant. This algorithm also has the disadvantage that for ensuring convergence a strict positive real condition (passivity) of the plant is needed.

Another aspect is the selection of a scheme for training the neural networks. Two different schemes are found in the literature. One method is to update the network weights after the entire batch of data has been processed. This is referred to as batch learning [36]. The other method is to update the weights of the neural network after each pattern has been processed. This is referred to as pattern learning [36]. The gradient descent method is exactly implemented by the batch learning method.

While the validity of pattern learning in a strict mathematical sense isn't available, it is seen to be practically effective. It has been shown [37] that pattern learning in the case of a feedforward network is valid for nonlinear activation networks provided that the learning rate is small.

In our work we will be dealing with the representation of nonlinear systems by Class IV and its variations using pattern learning and the series-parallel identification model.

4.2 Neural Network as Self-tuning Controller

One way of using neural networks in adaptive control systems is through the self-tuning principle. As described earlier, self tuning controllers are basically of two types, namely implicit and explicit.

A neural network can be used as an implicit self tuning controller provided that the controller dynamics can be directly learned by a neural network. Using this approach it is possible to determine the control input directly as the output of the neural network so that the plant output follows the desired set point. Schemes for using the neural network as self-tuning controllers- both implicit and explicit are shown in the figures 4.7 and 4.8.

Using a neural network as an explicit self-tuning controller is not optimum from a computational point of view. This is because of the extra computations involved. Hence we will be concerned with the utilization of neural network as implicit self-tuning controller.

The two nets shown in fig.4.8 are in fact equivalent with one being a mirror image of the other. The neural network, when used as an identifier, tries to approximate

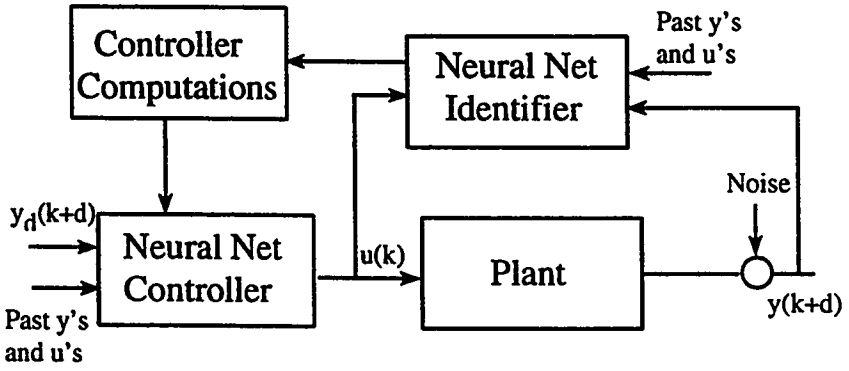


Figure 4.7: Neural Network As An Explicit Self-Tuning Controller

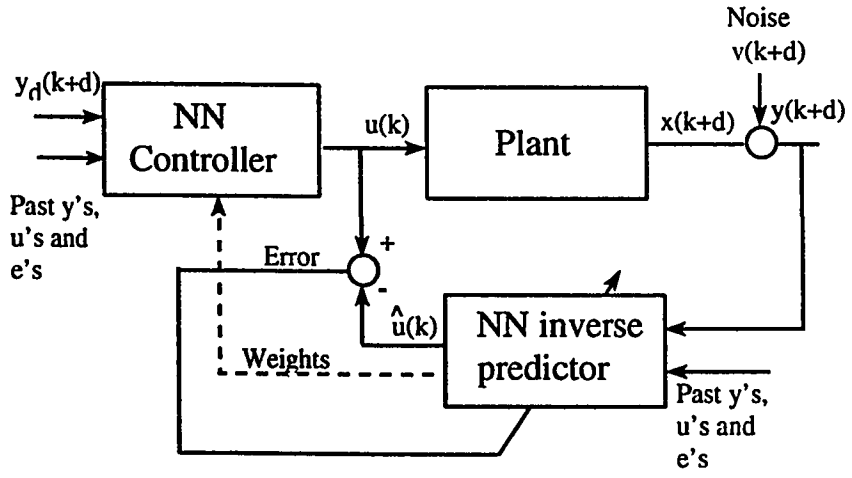


Figure 4.8: Neural Network As An Implicit Self-Tuning Controller

the inverse dynamics of the model. If the function f in equation 4.4 is known then a neural network *NN* which approximates f^{-1} over the area of interest can be designed off-line. But in adaptive control problems the function f is generally unknown and hence the neural network is to be trained on-line to estimate the function f^{-1} using the input-output pairs.

The other neural network in the forward path functions as the controller with its output being the actual plant input. The connection weights are adjusted such that the difference between the actual plant input and the estimated plant input (as determined by the neural network used as the identifier) is minimized. This learning has been called the direct learning architecture [38].

Denoting the network by *NN[y(k),u(k),W]* where y and u denote the output and the input at the k^{th} instant respectively, the estimation of f^{-1} reduces to a parameter estimation problem where the parameters to be estimated are the weights of the neural network.

In figure 4.8 the training error is given by

$$e(k+1) = u(k+1) - \hat{u}(k+1) \tag{4.5}$$

Generally the function to be minimized is the instantaneous squared error given by $J = \frac{1}{2}||e(k+1)||^2$. Using the well known delta rule [25] the weight update algorithm is given as

$$W(k+1) = W(k) - \eta_k \left[\frac{\partial^+ J}{\partial W}\right]^T_{W=W(k)} \tag{4.6}$$

$$= W(k) + \eta_k e(k+1).\left[\frac{\partial^+ \hat{u}}{\partial W}\right]_{W=W(k)} \tag{4.7}$$

where $\eta_k > 0$ is the learning rate parameter (also called the step size) and the plus sign indicates the ordered derivative [27].

The learning rate η_k is an important factor in the training of feedforward networks. While large values of η_k can lead to oscillation and instability, small values of η_k make the convergence very slow. Hence the backpropagation algorithm though enjoying the massively parallel processing capability of the network, suffers nevertheless from the drawback of slow convergence.

4.3 Implicit Self Tuning Control through minimum variance strategy

The theory of implicit self tuning control of inverse stable systems has been originally developed by Astrom and Wittenmark [4]. They have addressed linear systems only. In order to use this idea for implicit self tuning control of nonlinear systems, we have to restructure their algorithm. In order to gain insight into the working of the algorithms, we will first restructure the self-tuning algorithm for linear systems, both deterministic and stochastic cases.

4.3.1 Stochastic Linear plants

In the presence of noise the linear system can be modelled by an ARMAX representation as

$$y(k) + A(z^{-1})y(k-d) = B(z^{-1})u(k-d) + C(z^{-1})\bar{\xi}(k) \tag{4.8}$$

where u is the control variable, y is the output and $\bar{\xi}$ denotes the white noise. The delay is denoted by d. Define

$$A(z^{-1}) = a_0 + \cdots + a_{n_a} z^{-n_a} \tag{4.9}$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} + \cdots + b_{n_b} z^{-n_b} \tag{4.10}$$

$$b_0 \neq 0$$

$$C(z^{-1}) = 1 + c_1 z^{-1} + \cdots + c_{n_a} z^{-n_a} \tag{4.11}$$

Dividing throughout by b_0, transposing all terms of equation 4.8 except u(k-d) to the right we obtain the inverse dynamics of the plant as

$$u(k-d) = \frac{1}{b_0} y(k) + \bar{A}(z^{-1}) y(k-d) + \bar{B}(z^{-1}) u_a(k-d) + \bar{C}(z^{-1}) \xi(k) \tag{4.12}$$

where

$$\bar{A}(z^{-1}) = \frac{1}{b_0} A(z^{-1}) \tag{4.13}$$

$$\bar{B}(z^{-1}) = -(B(z^{-1}) - b_0)/b_0 \tag{4.14}$$

$$\bar{C}(z^{-1}) = -\frac{1}{b_0} C(z^{-1}) \tag{4.15}$$

Rewriting equation 4.12 such that all the future unknown terms are collected in a single term e(t) we have

$$u(k-d) = \frac{1}{b_0} y(k) + \bar{A}(z^{-1}) y(k-d) + \bar{B}(z^{-1}) u_a(k-d) + \bar{F}(z^{-1}) e(k-d) + e(k) \tag{4.16}$$

where

$$e(k) = \bar{c}_0 \xi(k) + \bar{c}_1 \xi(k-1) + \cdots + \bar{c}_{d-1} \xi(k-d+1) \tag{4.17}$$

$$\text{and} \quad \bar{F}(z^{-1}) = \frac{\bar{c}_d + \bar{c}_{d+1} z^{-1} + \cdots + \bar{c}_{n_a} z^{-n_a+d}}{\bar{c}_0 + \bar{c}_1 z^{-1} + \cdots + \bar{c}_{d-1} z^{-d+1}} \tag{4.18}$$

Equation 4.18 can be approximated as

$$\bar{F}(z^{-1}) = f_0 + f_1 z^{-1} + \cdots + f_m z^{-m} \tag{4.19}$$

using an adequate number of terms.

But another problem in the implementation of equation 4.16 is that the noise sequences appearing in the equations are also unknown. Hence we estimate them as

$$\hat{e}(k) = u(k-d) - \hat{u}(k-d)$$

where $\hat{u}(k-d)$ is the estimated input obtained as

$$\hat{u}(k-d) = \frac{1}{b_0}y(k) + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) + \bar{F}(z^{-1})\hat{e}(k-d) \tag{4.20}$$

Equation 4.20 can now be effectively used for estimating the parameters of $\bar{A}$, $\bar{B}$, $\bar{F}$, which are given following equations 4.12 and 4.16 and b_0.

Neural network controller

Although equation 4.20 can also be used in the controller,a problem arises from the fact that y(k)is unknown. However, a predicted value of y(k) based on the available information may be used. We let the desired value of the output y(k) predicted at time (k-d), denoted by $y(k|k-d)$ equal y_d. By predicting the noise terms after the $(k-d)^{th}$ instant equal to their mean, namely zero, we have

$$u(k-d) = \frac{1}{b_0}y_d + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) + \bar{F}(z^{-1})\hat{e}(k-d) \tag{4.21}$$

Using the estimates of the residuals, $\hat{e}$(t), equation 4.21 is used to obtain the actual control input to the plant. Hence

$$\hat{u}(k-d) = \frac{1}{b_0}y(k) + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) + \bar{F}(z^{-1})\hat{e}(k-d) \tag{4.22}$$

$$u(k-d) = \frac{1}{b_0}y_d(k) + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) + \bar{F}(z^{-1})\hat{e}(k-d) \tag{4.23}$$

The error $\hat{e}(k)$ is used to adjust the weights of the neural network. As a result of this adjustment the neural network controller output changes so as to cause the plant to follow the set point.

It has been shown in [28] that for a linear system it is not necessary to estimate the parameters corresponding to the noise terms. In that case the only difference is that $C(z^{-1})$ can be assumed as 1. As a result equations 4.22 and 4.23 can be written as

$$\hat{u}(k-d) = \frac{1}{b_0}y(k) + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) \tag{4.24}$$

$$u(k-d) = \frac{1}{b_0}y_d + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) \tag{4.25}$$

where, as earlier $u(k-d)$ denotes the output of the neural network controller and $\hat{u}(k-d)$ is the output of the inverse dynamics neural network. The control strategy that is adopted is the minimum variance control. Using this strategy, the closed loop poles consists of the zeros of the open loop plant as well as additional poles at the origin of the z-plane .

Similar analysis for linear deterministic plants is given in the appendix.

4.3.2 Stochastic nonlinear plants

In the case of linear systems only a single layered neural network is enough. But for nonlinear systems at least two layers are required. This is because a single layer is incapable of representing a nonlinear function adequately. A network with more than two layers is capable of representing a nonlinear function [34] [35]. A bias is also added to both the input and the hidden layers.

In the presence of additive noise we represent the nonlinear system as

$$\begin{aligned} y(k) &= f[y(k-d), y(k-d-1), \cdots, y(k-d-n_y); u(k-d), u(k-d-1), \cdots, \\ &\quad u(k-d-n_u); \xi(k-1), \xi(k-2), \cdots, \xi(k-d-n_y)] + \xi(k) \end{aligned} \tag{4.26}$$

where $\xi(k)$ is a white noise. Then the minimum variance strategy as defined in the earlier chapter is obtained by writing u(k-d) using an inverse function as

$$\begin{aligned} u(k-d) &= g[y(k), y(k-d), \cdots, y(k-d-n_y); u_a(k-d-1), \cdots, u_a(k-d-n_u);, \\ &\quad e(k-d), \cdots, e(k-d-n_y)] + e(k) \end{aligned} \tag{4.27}$$

where e(k) is given by equation 4.17 with c_i being $\partial f(.)/\partial \xi(k-i)$ [39] and $g = f^{-1}$. An estimate of the control input can be obtained from the equation 4.27 as

$$\begin{aligned} \hat{u}(k-d) &= g[y(k), y(k-d), \cdots, y(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u);, \\ &\quad \hat{e}(k-d), \cdots, \hat{e}(k-d-n_y)] \end{aligned} \tag{4.28}$$

The controller output is obtained analogously as in the linear case. Hence

$$\begin{aligned} u(k-d) &= g[y_d(k), y(k-d), \cdots, y(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u), \\ &\quad \hat{e}(k-d), \cdots, \hat{e}(k-d-n_y))] \end{aligned} \tag{4.29}$$

The unknown noise sequences in have been substituted by their estimates as in the case of the linear systems. The estimates are obtained as

$$\hat{e}(k) = u(k-d) - \hat{u}(k-d) \tag{4.30}$$

and use these estimates as the inputs to both the controller and the inverse dynamics neural network. The estimated error $\hat{e}(k)$ is used to adjust the weights of the neural network using the backpropagation algorithm.

But it has also been seen through simulations that in many cases it is not necessary to estimate the parameters corresponding to noise even for nonlinear systems. The behavior of the controlled system in both cases are found quite similar.

Disadvantage of minimum variance control

The above strategy of minimum variance control has some demerits namely

1. It is incapable of controlling an inverse unstable system because the controller explicitly attempts to cancel the forward path zeros which may be outside the unit circle.
2. the control inputs produced may be excessive.
3. the only modification that can be done by the user is to vary the sample time.
4. requires accurate knowledge of the time delay in the plant.

In case of the inverse unstable plants, the minimum variance controller produces unbounded control input. In addition, the variation of the estimated parameters from the true parameters causes both the input and the output to be unbounded.

To overcome these disadvantages and to allow for a wider range of control objectives the self tuning controller based on other strategies are also developed [7],[8],[9]. The methods minimize the variance of an auxiliary output $\phi(k)$ given by

$$\phi(k) = P(z^{-1})y(k) + Q(z^{-1})u(k-d) - R(z^{-1})y_d(k-d) \tag{4.31}$$

where $P(z^{-1})$, $Q(z^{-1})$ and $R(z^{-1})$ are polynomials in z^{-1}.

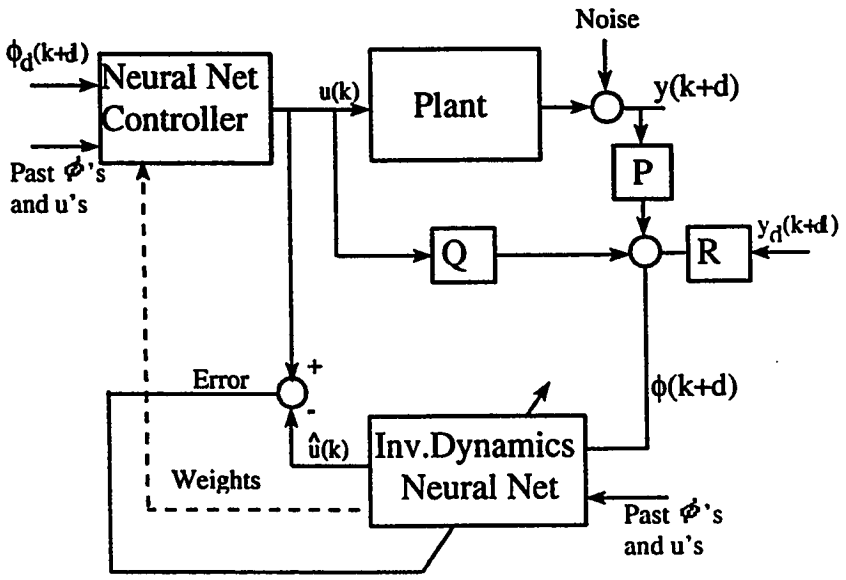


Figure 4.9: Self Tuning Control of an inverse unstable linear system using neural network

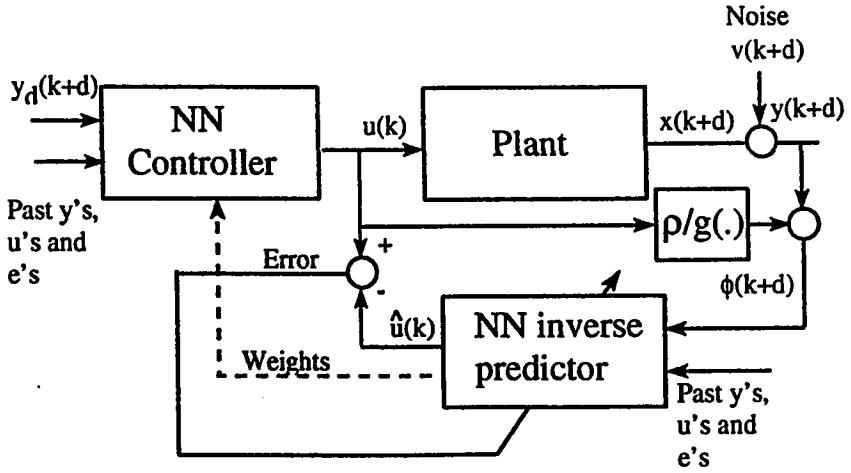


Figure 4.10: Implementation diagram of strategy II

4.4 Implicit self tuning control through the generalized minimum variance strategy

A linear nonminimum phase or inverse unstable discrete time system is one with zeros outside the unit circle. On the other hand a nonlinear inverse unstable system has, as the name indicates, an unstable inverse dynamics. As earlier we will first deal with linear systems.

4.4.1 Stochastic linear plants

In the presence of noise the linear system can be modelled as

$$\ddot{A}(z^{-1})y(k) = B(z^{-1})u(k-d) + C(z^{-1})\xi(k) \tag{4.32}$$

where ξ denotes a white noise and

$$\ddot{A}(z^{-1}) = 1 + z^{-d}A(z^{-1}) = 1 + a_0 z^{-d} + \cdots + a_{n_a} z^{-d-n_a} \tag{4.33}$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} + \cdots + b_{n_b} z^{-n_b} \tag{4.34}$$

$$b_0 \neq 0$$

$$C(z^{-1}) = 1 + c_1 z^{-1} + \cdots + c_{n_a} z^{-d-n_a} \tag{4.35}$$

where one or more roots of B may be outside the unit circle. Let the pseudo output be given by

$$\phi(k) = Py(k) + Qu(k-d) \tag{4.36}$$

Letting $Q = \lambda$ and P =1 we have

$$\phi(k) = y(k) + \lambda u(k-d) \tag{4.37}$$

From equations 4.32 and 4.37 we have

$$\ddot{A}(z^{-1})\phi(k) = [B(z^{-1}) + \lambda\ddot{A}(z^{-1})]u(k-d) + C(z^{-1})\xi(k) \quad (4.38)$$

$$\text{or } \ddot{A}(z^{-1})\phi(k) = \tilde{B}(z^{-1})u(k-d) + C(z^{-1})\xi(k) \quad (4.39)$$

$$\text{where } \tilde{B} = [B(z^{-1}) + \lambda\ddot{A}(z^{-1})] \quad (4.40)$$

Dividing throughout by $\tilde{b}_0$, where $\tilde{b}_0 = b_0 + \lambda$ transposing all terms except u(k-d) to the right we have as in equation 4.12

$$u(k-d) = \frac{1}{\tilde{b}_0}[\ddot{A}(z^{-1})\phi(k) + \bar{B}(z^{-1})u_a(k-d) + \bar{C}(z^{-1})\xi(k)] \quad (4.41)$$

where

$$\bar{B} = -(\tilde{B}(z^{-1}) - \tilde{b}_0) \quad (4.42)$$

$$\bar{C} = -C(z^{-1}) \quad (4.43)$$

The above equation for the inverse dynamics can be written as

$$u(k-d) = \frac{1}{\tilde{b}_0}[\phi(k) + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u_a(k-d) + F(z^{-1})e(k-d) + e(k)] \quad (4.44)$$

where e(k) and $F(z^{-1})$ are defined in equation 4.17 and 4.18 respectively. The equation to estimate the control input can then be obtained by estimating the residuals.

Neural network controller

In an analogous way as in equation 4.21, we may write

$$u(k-d) = \frac{1}{\tilde{b}_0}[\phi_d + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u(k-d) + F(z^{-1})\hat{e}(t-d)] \quad (4.45)$$

But a problem with the implementation of the above controller equation is that we do not have information about ϕ_d. To overcome this problem we substitute 4.37 in 4.45 to get

$$\begin{aligned} u(k-d) &= \frac{\tilde{b}_0}{\tilde{b}_0 - \lambda}\frac{1}{\tilde{b}_0}[y_d + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u(k-d) \\ &\quad + F(z^{-1})\hat{e}(t-d)] \end{aligned} \tag{4.46}$$

The noise sequence is estimated as

$$\hat{e}(k) = u(k-d) - \hat{u}(k-d) \tag{4.47}$$

When these estimates are used, the equations for the identifier and controller neural networks respectively become

$$\begin{aligned} \hat{u}(k-d) &= \frac{1}{\tilde{b}_0}[\phi(k) + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u(k-d) \\ &\quad + F(z^{-1})\hat{e}(k-d)] \end{aligned} \tag{4.48}$$

$$\begin{aligned} u(k-d) &= \frac{\tilde{b}_0}{\tilde{b}_0 - \lambda}\frac{1}{\tilde{b}_0}[y_d + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u(k-d) \\ &\quad + F(z^{-1})\hat{e}(k-d)] \end{aligned} \tag{4.49}$$

As it has been seen earlier that in linear systems, $C(z^{-1})$ may be set to 1. In that case as earlier we have

$$\hat{u}(k-d) = \frac{1}{\tilde{b}_0}[\phi(k) + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u(k-d)] \tag{4.50}$$

$$u(k-d) = \frac{\tilde{b}_0}{\tilde{b}_0 - \lambda}\frac{1}{\tilde{b}_0}[y_d + A(z^{-1})\phi(k-d) + \bar{B}(z^{-1})u(k-d)] \tag{4.51}$$

Using the above strategy, once the estimated parameters converge to their true values, y(k) will approach $y_d(k)$. It can be shown that this approach is equivalent to minimizing $||y_d - y(k)||^2 + \lambda||u(k-d)||^2$ [40]. Further, a simple root locus analysis

shows that this scheme is capable of producing a stable control system for all open loop unstable minimum phase and all open loop stable nonminimal phase plants. The setpoint tracking in this approach is based on feedforward gain. Thus errors in estimation of the inverse dynamics may produce an offset.

In order to ensure proper set point following, even in the case of inaccuracies in identification we adopt a different strategy. Let us call it strategy II. In this case we assume the pseudo-output as

$$\bar{\phi}(k) = y(k) + \lambda[u(k-d) - u(k-d-1)] \tag{4.52}$$

This is a GMV strategy that uses incremental control which is known to have the advantages of automatic steady-state reference point tracking as well as the removal of constant disturbances. Hence one equivalently gets

$$\begin{aligned} \ddot{A}(z^{-1})\bar{\phi}(k) &= [B(z^{-1}) + \lambda(1-z^{-1})\ddot{A}(z^{-1})]u(k-d) + C(z^{-1})\xi(k) && (4.53) \\ &= \bar{B}u(k-d) + C(z^{-1})\xi(k) && (4.54) \end{aligned}$$

With this strategy, in case of constant setpoint, upon parameter convergence u(k) will become constant making u(k-d) equal to u(k-d-1). Thus, as $\bar{\phi}_d(k)$ approaches $y_d(k)$, y(k) also approaches $y_d(k)$ as can be seen from equation 4.52. This scheme can be shown to be equivalent to minimizing $||y_d - y(k)||^2 + \lambda||u(k-d) - u(k-d-1)||^2$ [40]. The above approach based on strategy II is in fact equivalent to the introduction of an integrator in the forward path. The implementation diagram for strategy II is shown in figure 4.10.

While the latter approach does ensure set point following, it is not capable of producing a stable control system for all open loop stable nonminimal phase plants. In order to understand why this is so we will have to look into the root locus plots.

We assume that the plant is exponentially stable but inverse unstable.

We are interested in the polynomials $\bar{B}$ and $\tilde{B}$ which correspond to the denominator of the controller. $\tilde{B}$ and $\bar{B}$ in turn depend on A and B. Figure 4.11 shows typical root loci of $\tilde{B}$ and $\bar{B}$ as λ increases. The cross (x) corresponds to the roots of B which are outside the unit circle. The small circle, inside the unit circle corresponds to the roots of A which is stable. In figure 4.11, it is assumed that the unstable root of B is real.

In the case of figure 4.11(i) which corresponds to the strategy I it can be seen that a stable control system is realizable. This is because there does exist a value of λ for which the roots of $\bar{B}$ can be driven inside the unit circle. This is not so in the case of strategy II. In this case as can be seen from figure 4.11(ii), that there exists no value of λ for which the controller can be stabilized. The problem arises from the fact that the $(1 - z^{-1})$ term in equation 4.53 introduces additional zero at z=1.

Figures corresponding to the cases whereby B has complex roots are also shown in figure4.12. In all the other cases there does exist a value of λ for which the controller can be stabilized for both strategies. Thus it can be seen that for both of these strategies there are some associated problems.

4.4.2 Stochastic nonlinear plants

We assume that the plant is of the following type

$$\begin{aligned} y(k) &= f[y(k-d),\cdots,y(k-d-n_y);u(k-d),\cdots,u(k-d-n_u); \\ &\quad \xi(k-1),\cdots,\xi(k-d-n_y)] + \xi(k) \end{aligned} \tag{4.55}$$

After rearranging the noise terms such that all the future unknown terms in ξ are collected together in a single term e(k), which is given by 4.18 we would obtain the

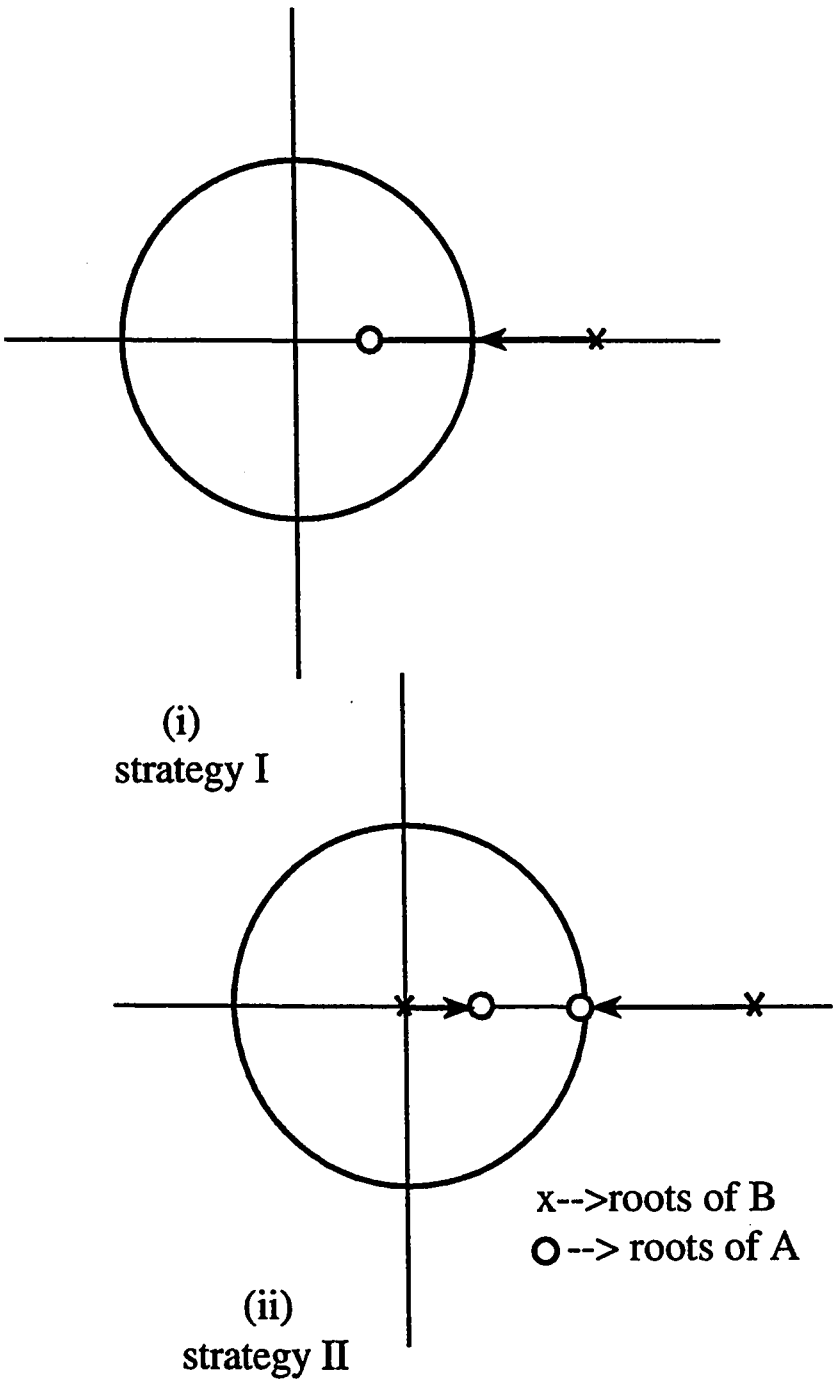


Figure 4.11: Root locus plots for strategy I and II with real roots of A & B

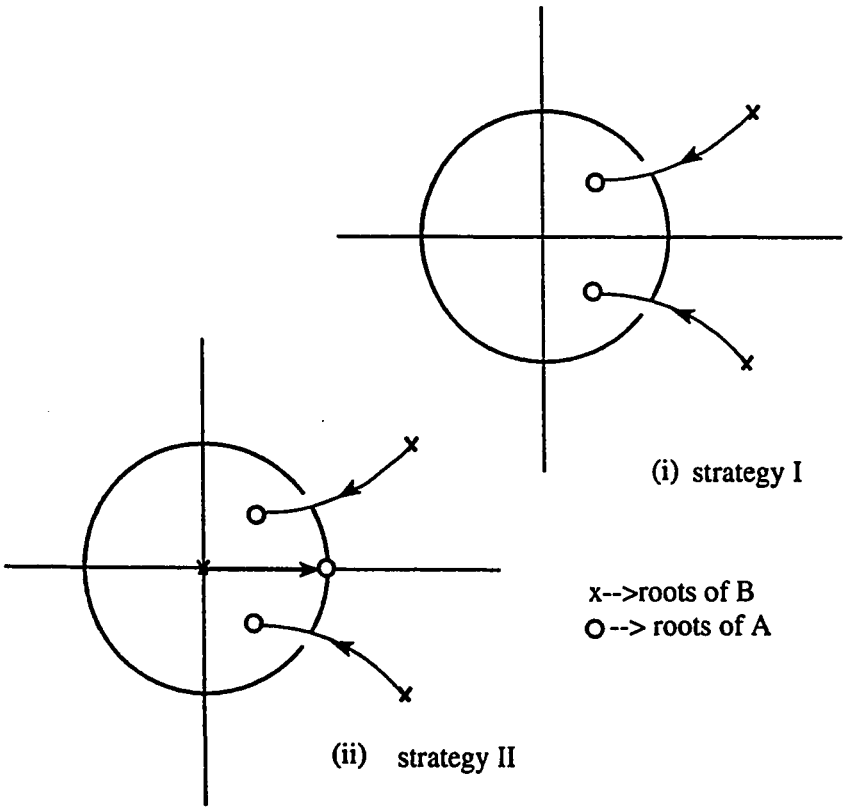


Figure 4.12: Root locus plots for strategy I and II with complex roots of A & B

inverse dynamics model as

$$\begin{aligned}\hat{u}(k-d) &= g[y(k), y(k-d), \cdots, y(k-d-n_y); u_a(k-d-1), \cdots, u_a(k-d-n_u); \\ & \quad e(k-d), \cdots, e(k-d-n_y)] + e(k) \end{aligned} \tag{4.56}$$

But for an inverse unstable system, the minimum variance strategy will produce unbounded control signal. In addition it will be extremely sensitive to the difference between the actual parameters and the estimated parameters. Hence, as in the case of the linear systems, we minimize the variance of the pseudo output. Using the approach of [41] and [42] it can be proved that the pseudo output in case of a nonlinear function has to have the form (see appendix C)

$$\phi(k) = Py(k) + \frac{\rho}{\hat{h}} u(k-d) \tag{4.57}$$

where P is assumed as 1 and $\hat{h}$ is to be proportional to the gradient of the output of the inverse dynamics neural network with respect to the pseudo output. Otherwise, the estimated controller does not minimize the variance of the pseudo output. Hence $h(t)$ can be estimated as Hence

$$\hat{h}(.) = \frac{\partial \hat{u}(k-d))}{\partial \phi(k))} \tag{4.58}$$

In order to estimate $\hat{h}(.)$, we again make use of the neural network backpropagating a 1 instead of the error. The term $\hat{h}(.)$ is obtained at the node corresponding to $\phi(k)$. Thus it is possible to calculate the pseudo-outputs. Hence the estimate of the control input to the plant can be obtained from

$$\begin{aligned}\hat{u}(k-d) &= \tilde{g}[\phi(k), \phi(k-d), \cdots, \phi(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u); \\ & \quad \hat{e}(k-d), \cdots, \hat{e}(k-d-n_y)] \end{aligned} \tag{4.59}$$

where $\tilde{g}$ is a new function resulting from a change of the argument from y to ϕ.

Neural Network Controller for strategy I

As earlier the function of the neural network controller is to determine the input to the plant such that the actual output of the plant follows the desired output. The equation of the controller is given by

$$\begin{aligned} u(k-d) &= \tilde{g}[\phi_d, \cdots, \phi(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u); \\ &\quad \hat{e}(k-d), \cdots, \hat{e}(k-d-n_y)] \end{aligned} \tag{4.60}$$

The generalized minimum variance strategy attempts to equate $\phi(t)$ to ϕ_d. However, since ϕ_d is unknown, if ϕ_d is replaced by y_d, the resulting control produces an offset in the output.

In this approach the calculated input is scaled using a first order approximation to $\tilde{g}(.)$ in $\phi_d(k)$. The derivation applies for small ρ. Using (4.60), the first order approximation can be obtained as

$$\tilde{g}[\phi_d, \cdots,] = \tilde{g}[y_d, \ldots] + \frac{\partial \tilde{g}[y_d, \cdots,]}{\partial y_d} \frac{\rho u(t-d)}{g(t)} \tag{4.61}$$

Further, since $\partial \tilde{g}(.)/\partial u(k) = 0$ and $\partial y_d/\partial \phi_d = 1$, it follows from (4.57), (4.58) and (4.60) that

$$g(k) = \frac{\partial^+ \hat{u}(k-d)}{\partial \phi(k)} = \frac{\partial \tilde{g}[y_d, \cdots,]}{\partial \phi_d(k)} \tag{4.62}$$

$$= \frac{\partial \tilde{g}[y_d, \cdots,]}{\partial y_d} \frac{\partial y_d}{\partial \phi_d} = \frac{\partial \tilde{g}[y_d, \cdots,]}{\partial y_d} \tag{4.63}$$

Combining (4.60),(4.61) and (4.63), one gets

$$\begin{aligned} u(k-d) &= \frac{1}{1-\rho} \tilde{g}[y_d(k), \phi(t) \cdots, \phi(k-d-n_u); u(k-d-1), \cdots, u(k-d-n_u), \\ &\quad \hat{e}(k-d), \cdots, \hat{e}(k-n_y-d)] \end{aligned} \tag{4.64}$$

This approach is similar to the strategy I used in the linear case. The setpoint tracking is based on a feedforward compensation. Thus error in estimation in the inverse dynamics model may effect the performance. In addition, the method based on a first order approximation of a nonlinear relationship may not completely eliminate the offset problem. Hence analogous to the linear case we use strategy II. As remarked earlier while strategy II is capable of completely eliminating the offset, it is incapable of controlling some plants.

Strategy II

In this approach, $\phi(k)$ is modified by

$$\phi^{'}(k) = y(k) + [\rho/h(k)][u(k-d) - u(k-d-1)] \tag{4.65}$$

This strategy is equivalent to the use of an integrator as pointed out earlier. It can be observed from (4.65), that at steady state when the input reaches constant value, $\phi^{'}(k)$ becomes y(k) and therefore use of y_d for $\phi^{'}_d$ does not effect the steady state performance. The same scheme as shown in figure 4.10.

As earlier the estimate of the control input can be obtained as

$$\begin{aligned}\hat{u}(k-d) = \underline{g}[&\phi^{'}(k), \phi^{'}(k-d), \cdots, \phi^{'}(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u); \\ &\hat{e}(k-d), \cdots, \hat{e}(k-d-n_y)]\end{aligned} \tag{4.66}$$

Neural network Controller for strategy II

The equation of the controller is given as

$$\begin{aligned}u(k-d) = \underline{g}[&y_(d), \phi^{'}(k-d), \cdots, \phi^{'}(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u); \\ &\hat{e}(k-d), \cdots, \hat{e}(k-d-n_y)]\end{aligned} \tag{4.67}$$

The difference between the actual input to the plant as determined by the controller and the input estimated by the inverse dynamics neural network generates the error $e(k)$. This error is in turn used to change the weights of the neural network.

It is pointed out earlier that in nonlinear plants it may not be necessary to feed the noise terms $\hat{e}(k)$ to the neural networks. This point will be brought in the next chapter while reporting the simulation studies. The simulation results for different systems covering all the above categories are provided in the next chapter.

Chapter 5

SIMULATION RESULTS

SUMMARY: Simulation results with neural network controllers based on the different schemes outlined earlier have been provided. The heuristics used in order to overcome any implementation problems have also been given.

5.1 Introduction

In this chapter the simulation results for the various types of neural network controllers discussed before are presented. While the emphasis is on nonlinear plants, we will start the simulations with examples of linear plants. This is so as to get a theoretical understanding of the behavior of the controller which is easier to grasp in the case of linear plants.

5.2 Basis of Comparison

We will concentrate here on the performance of the proposed methods in terms of set-point tracking, stability and the convergence rates. In addition to this we will also look at parameter estimation for linear systems. In the case of a linear system, the neural network controller becomes a single layer and hence the estimated weights become identical to the estimated parameters. This is not so in the case of a nonlinear system where the controller becomes multilayered. Therefore the estimated network weights cannot be evaluated through direct comparison with any known parameter values.

5.3 Simulation Results

The disturbance to the system is generated using a Gaussian distribution of zero mean and variance σ denoted as $N(0, \sigma)$. Initial values of the neural network weights are all set to small random quantities. This is so as to prevent the network weights from being paralysed. Paralysis of the weights occurs when the weights get stuck at high values and hence the network doesn't learn at this stage. The values of the estimated parameters as well as the system output and input are all stored in separate data files. These are later plotted using the Adobe Illustrator package available on a Next Workstation. All the simulation work was done on the Next Workstation with the programs being written in C.

Only the conclusive results of typical systems are presented and analysed. No assumptions are made on the structure of the plant, though it is assumed that the orders of the plant and noise dynamics are known. This is required to determine

the number of input nodes in the neural network.

The existence of suitable inverse operators is also assumed. It is also assumed that weight matrices of the neural network exist such that the model gives the exact inverse of the plant. Through simulation study we found that two layered networks using the hyperbolic tangent activation function are adequate to learn the inverse dynamics of the simulated plants. Therefore in all our simulations, we have fixed the number of neural layers to 2, with one hidden and one output layer. The number of neurons in the input and the output layers are fixed depending on the plant orders and the number of plant outputs respectively. But the number of neurons in the hidden layer has to be found using simulations. This is because there are no rules which relate the nonlinearity to be modelled to the number of neurons in the hidden layer. Yet, this problem is not as acute as it sounds. This is because as a rule of thumb, which is established through simulations, in control applications generally the number of neurons in the hidden layer are nearly equal to the number of neurons in the input layer.

As far as linear plants are concerned, it is quite obvious that neural networks with no hidden layers and linear activation functions are enough to approximate them. Further, the number of neurons in both the input and the output layers is directly obtained from the plant orders and the number of plant outputs.

5.4 Minimum variance strategy

5.4.1 Stochastic linear plants

The plant in this case is described by the difference equation

$$y(k) - 1.6y(k-1) + 0.63y(k-2) = u(k-1) - 0.5u(k-2) + \xi(k) + 0.5\xi(k-1) \quad (5.1)$$

A single layered neural network is used to learn the inverse dynamics of the plant. The activation function of the different neurons is also taken as unity. The initial weight settings have been small. It is observed that the different weight settings do converge asymptotically to the plant parameters as expected.

The desired set point was varied between +5 and -5 after every 50 iterations. The minimum variance control strategy has been used. Simulations have been done using both a constant learning rate and variable scalar learning rate in the backpropagation algorithm. The weights were updated at every instant of time. The variable scalar learning rate is given by

$$\eta(k) = \frac{s(k)\mu}{l + s(k).||\Gamma(k)||^2} \quad (5.2)$$

$$0 \leq \mu \leq 2 \quad (5.3)$$

with s(k) being updated after each iteration as

$$s(k+1) = s(k)\frac{1 - (2-\mu)\eta(k)||\Gamma(k)||^2}{l} \quad (5.4)$$

where $\Gamma(k)$ is the gradient vector calculated through backpropagation in the k^{th} iteration and l is the number of weights in the neural network.

The learning is observed to be distinctly better using the variable scalar learning rate as compared with the constant learning rate. Suitable values of μ and s(0) are determined through simulations. The suitable value of s(0) has been found to be between $0.1 \sim 1$ while a suitable value of μ is about 1.98.

The control using the above formula is shown in figure 5.1. The plant performance is quite satisfactory especially considering that the standard deviation of the noise is 0.25. The overshoot at switchover though, is a little high, typically reaching about ± 5.5. But it has been observed that as k increases the overshoot reduces. Another point to be noted here is that the setpoint is also being flipped quite rapidly.

All the above simulations are done such that the parameters corresponding to the noise terms are also identified. But as indicated in the earlier chapter, it is not necessary to identify the noise parameters in the linear plants.

Simulations are also carried out without estimating the noise parameters. The resulting plant output is shown in figure 5.1b. A comparison of figures 5.1a and 5.1b shows similar steady state performance of the two. However the initial convergence of the former has been found to be better, with the choice of C polynomial affecting only the transients. However, the reduction of dimensionality in the latter due to elimination of noise parameters adds to the computational simplicity. A further point to be noted in all the above simulations is the ringing present in the input signal. This is primarily due to the bad location of the zeros of the plant, which the minimum variance control strategy attempted to cancel.

The above technique of minimum variance control is however incapable of controlling plants which are inverse unstable. This was also verified through simulations. As noted in the previous chapter that a minimum variance control attempts to can-

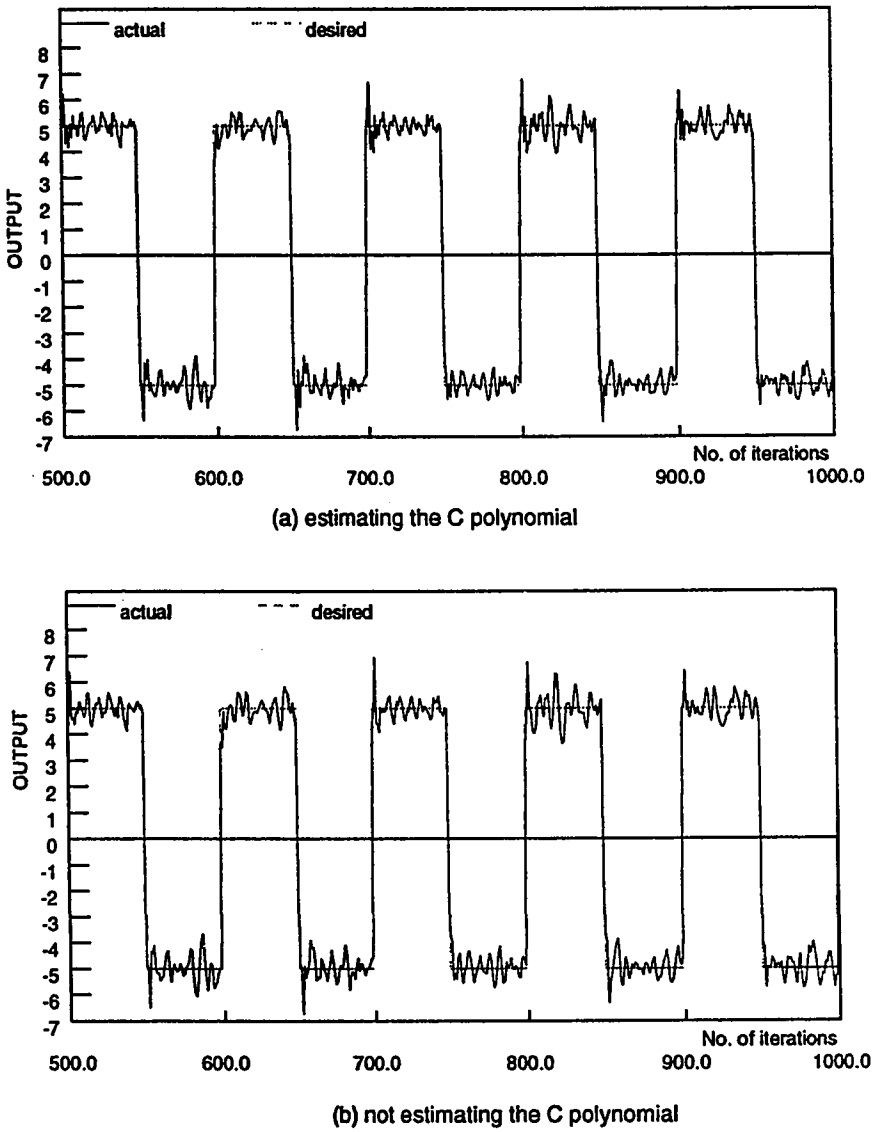


Figure 5.1: Stochastic linear plant using the minimum variance strategy and variable step size

cel the plant zeros, which for a inverse unstable plant, produces unbounded control input. It is also observed that if the initial weights are set to values near the true weights, although the input becomes unbounded, the neural network is still capable of learning the inverse plant dynamics of a nonminimal phase plant.

Since we are using a single layered neural network to learn the inverse dynamics of a linear plant the true weights of the neural network correspond to the parameters of the plant. It was observed that the weights of the neural network do asymptotically approach the parameters of the plant irrespective of the initial weight settings. The only care to be taken is that the initial weight settings be small enough.

5.4.2 Deterministic nonlinear plants

Making a slight digression from the order of the presentation in the earlier chapter, we will look at the simulation results using the minimum variance strategy with the neural network controller on a nonlinear plant. This is mainly in order to focus upon the tremendous improvement in the learning rates of the neural network using the proposed gain formulae. This is very obvious in the deterministic case. Simulation results on other plants have been provided in the appendix B.

The plant described by a NARX model, simulation results on which have been shown in this section is

$$\text{Plant I} \qquad y(k) = \frac{-0.9y(k-1) + u(k-1)}{1 + y(k-1)^2} \tag{5.5}$$

This model is taken from Su et al (1992). Since the plant is nonlinear, a two-layered neural network has been used. The nonlinear activation unit is taken as the hyperbolic tangent. One biased neuron is used in each of the input and the hidden layers. The parameters of the neural network are adjusted using the static

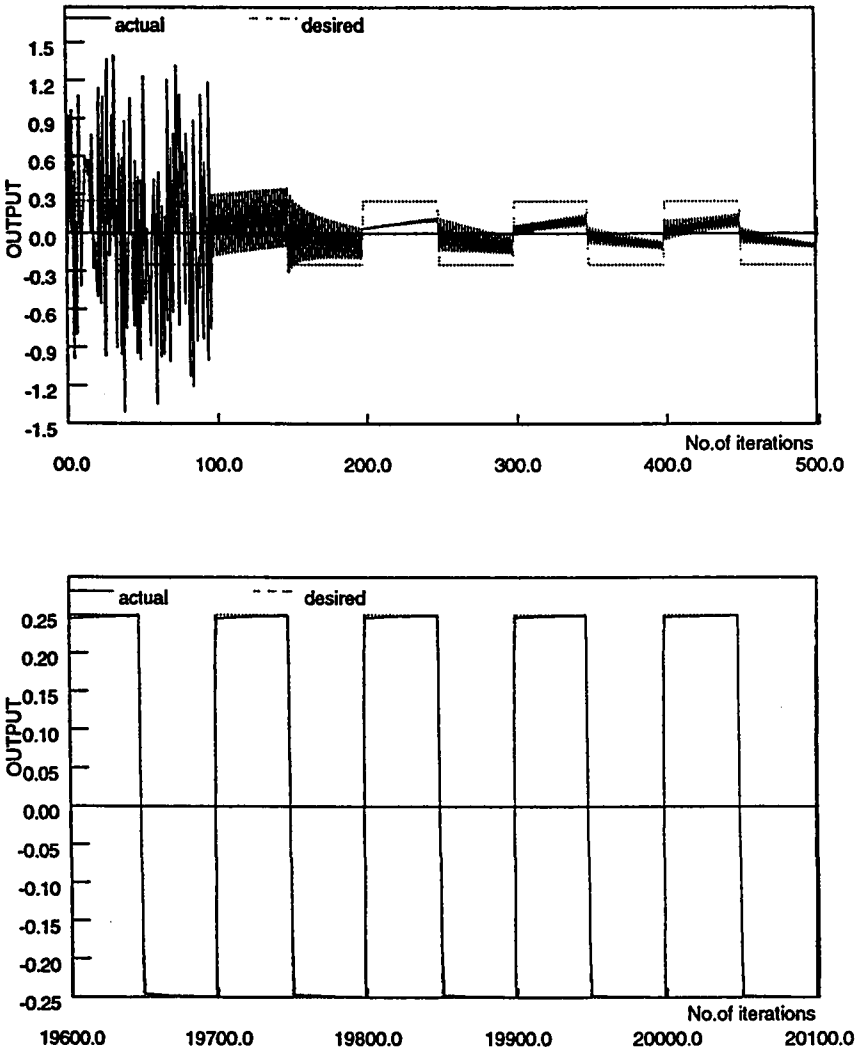


Figure 5.2: Deterministic NARX model using constant step size

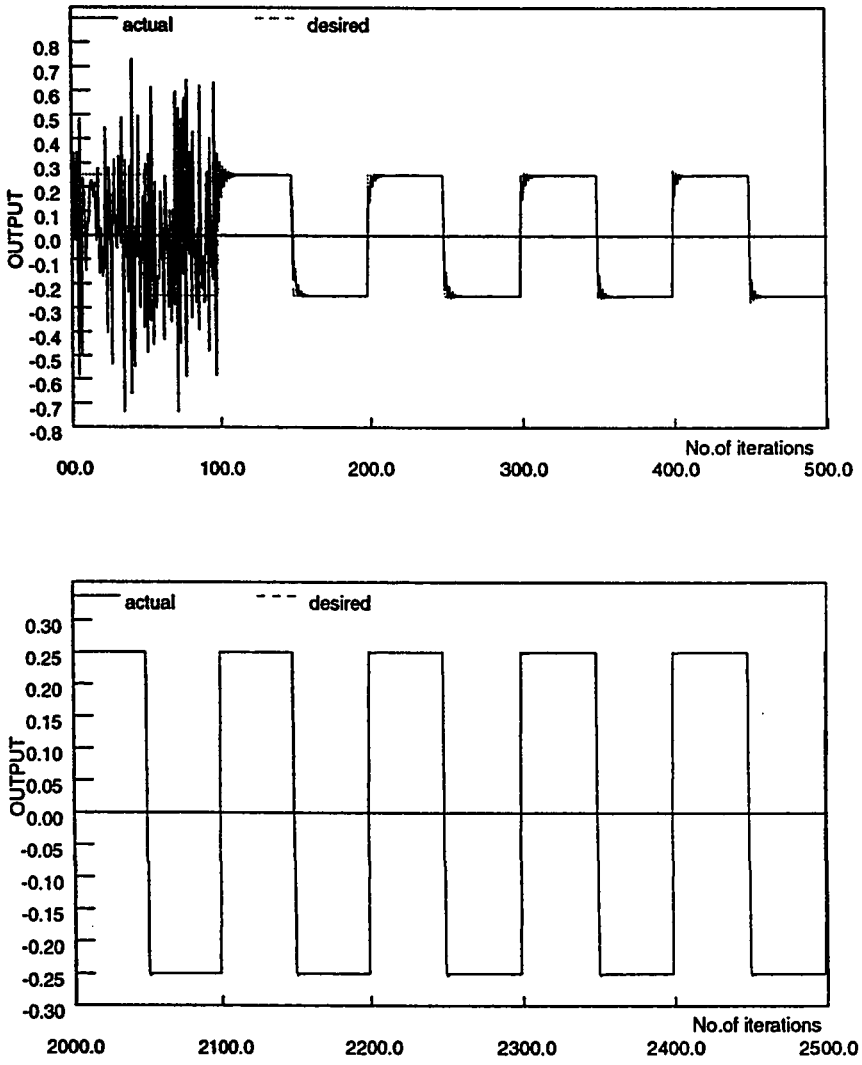


Figure 5.3: Deterministic NARX model using the variable step size

backpropagation method and the variable scalar learning rate as given in equation A.16.

In the case of Plant I the set point is switched between 0.25 and -0.25 after every 50 iterations. The control strategy used has been the minimum variance one. Dither is added to the input for the first 100 iterations in order to facilitate rapid learning. Simulations are performed using both the constant learning rate and the variable scalar learning rate. Figure 5.2 shows the plant performance when a constant step size of 0.1 is used to train the neural network controller. As can be seen the neural network controller takes quite a long time - about 20000 iterations to learn the plant dynamics. The performance of the same plant using the variable scalar learning rate to update the weights is given in figure 5.3. The plant dynamics are identified by the plant within about 100 iterations. Thus the drastic improvement in learning by the neural network with the variable scalar learning rate can be clearly seen from figures 5.2 and 5.3.

The earlier simulation study on nonlinear plants has been performed using the variable scalar learning rate given by equation A:16. A variable matrix learning rate has also been proposed in [43]. Simulation results using the matrix learning for the Plant I have been presented in figure 5.4. As earlier the set point varies between +0.25 and -0.25 while dither was added to the input for the first 100 iterations. The overshoot at the change of set point, which in our simulations has been very frequent, is quite negligible, typically about 0.03 to 0.04. Even this reduces with passage of time because of the improvement in the neural network learning. The plant then settles down at the desired command input. Thus it is seen that the learning rate is very fast, but the complexity of this formula is somewhat involved. Hence

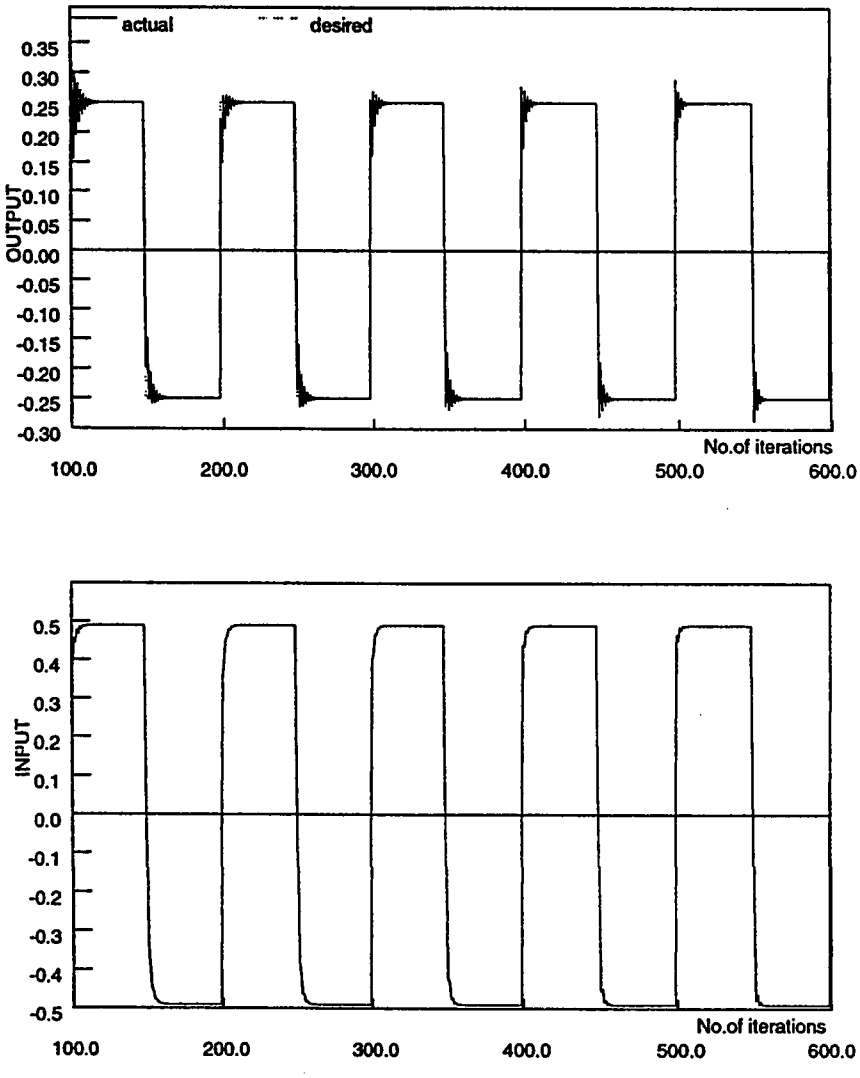


Figure 5.4: Deterministic NARX model using the matrix gain formula

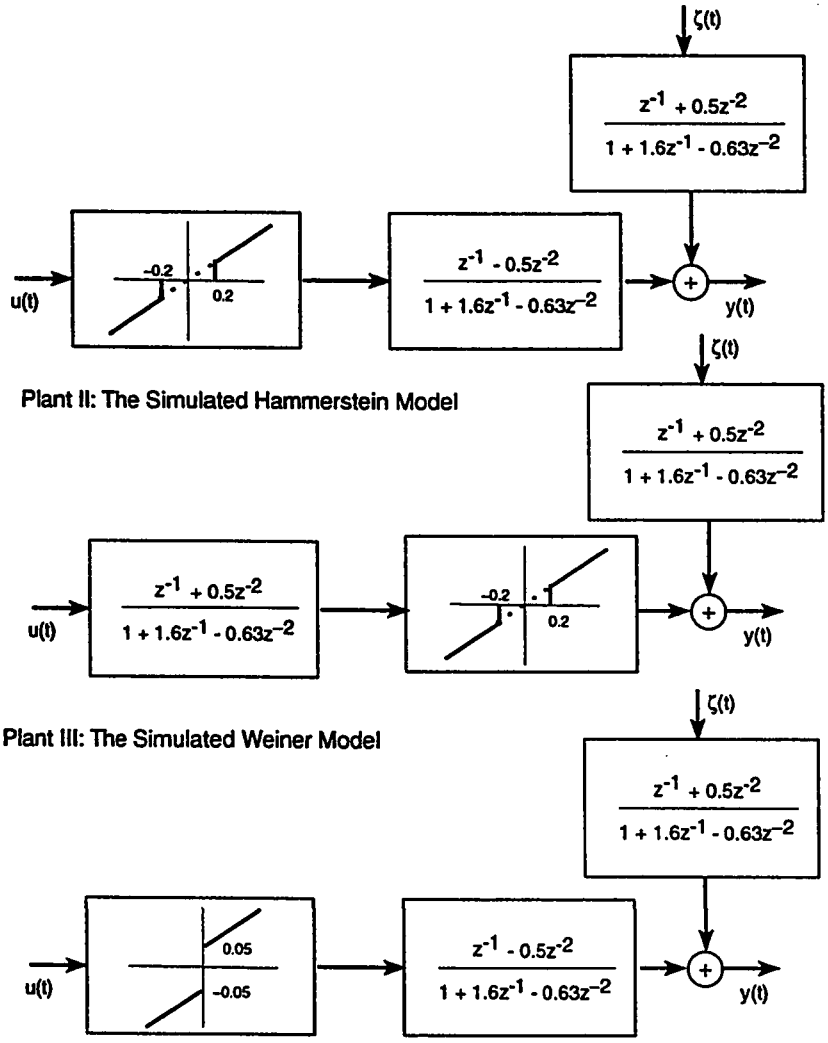


Figure 5.5: Nonlinear plant models

this formula can be used when the computational facility allows implementation of complex algorithms.

5.4.3 Stochastic nonlinear plants

Different nonlinear models are also simulated with the presence of noise. A two-layered neural network has been used. The configuration of the noisy plants represented as Hammerstein and Weiner models is shown in figure 5.5. Another model described by the following equations is also considered

$$\text{Modified Plant I:} \quad x(k) = \frac{-0.9x(k-1) + u(k-1)}{1 + x^2(k-1)} \tag{5.6}$$

$$y(k) = x(k) + v(k) \tag{5.7}$$

$$v(k) = \xi(k) + 0.5\xi(k-1) \tag{5.8}$$

This model is obtained by adding noise to the deterministic plant considered earlier (see 5.5). The step response of the plant is very oscillatory for low amplitude input and shows limit cycle oscillation for input amplitude greater than 0.6.

The set point is switched between ± 0.25 after every 50 iterations, 4 neurons are used in the hidden layer. Uniformly distributed random numbers ranging between -0.5 and +0.5 have been used as the dither input during the first 100 iterations to facilitate rapid learning of the neural network. The minimum variance control of the plant is shown in figure 5.6.

The nonlinearity represents a typical dead zone. For the Hammerstein model 6 neurons are used in the hidden layer. The setpoint is varied between ± 1. Dither is added for the first 100 iterations. The simulation results using the variable scalar learning rate for the Hammerstein model are shown in figure 5.7.

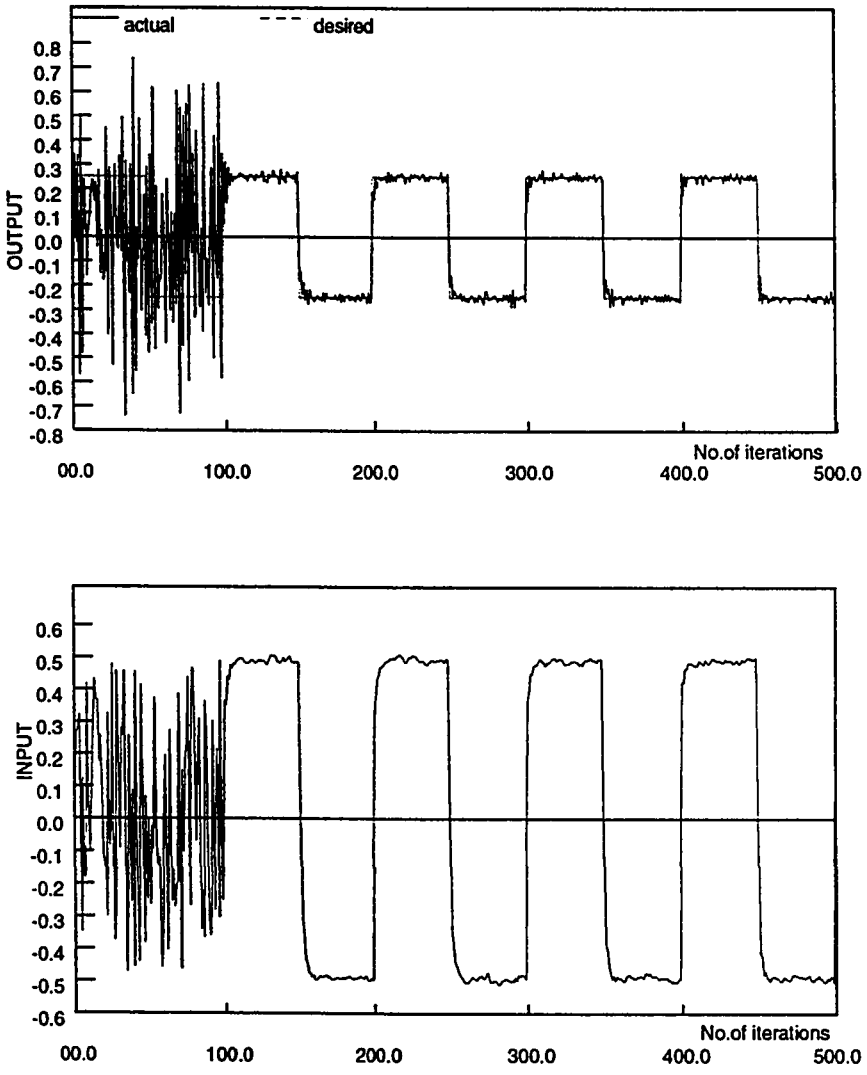


Figure 5.6: Minimum variance control of the stochastic NARX model

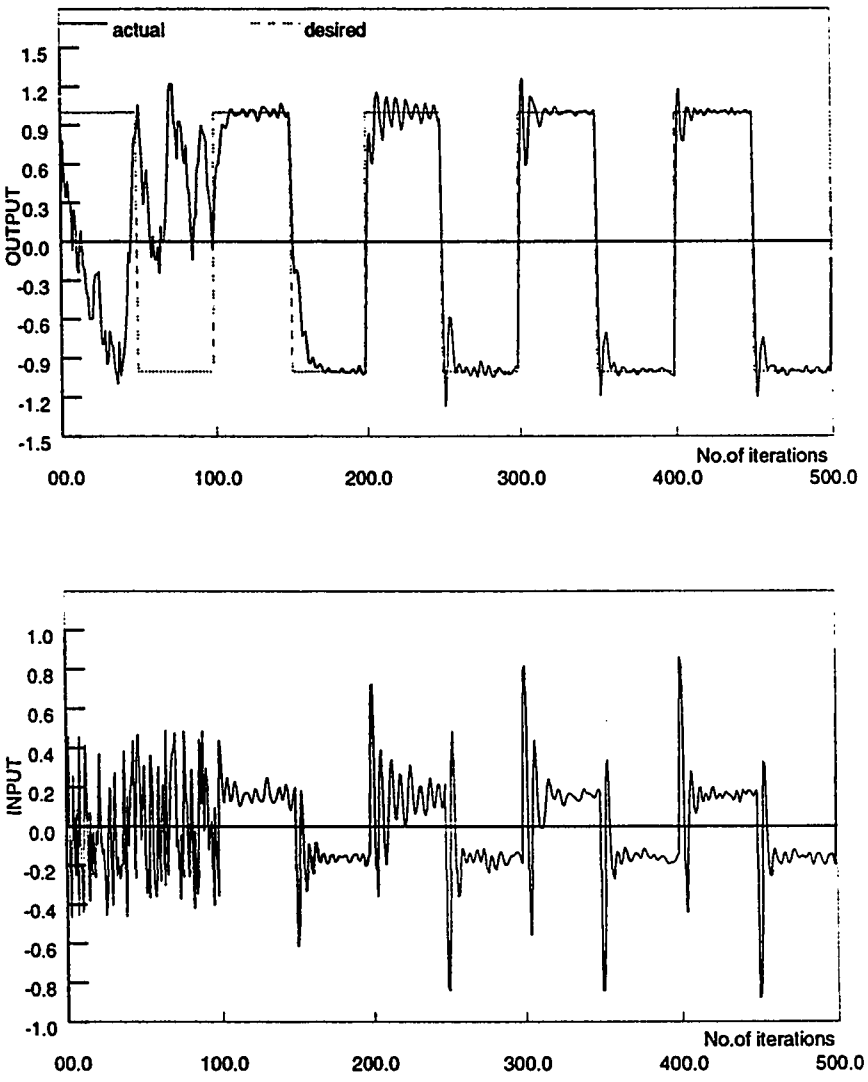


Figure 5.7: Minimum variance control of the stochastic Hammerstein model

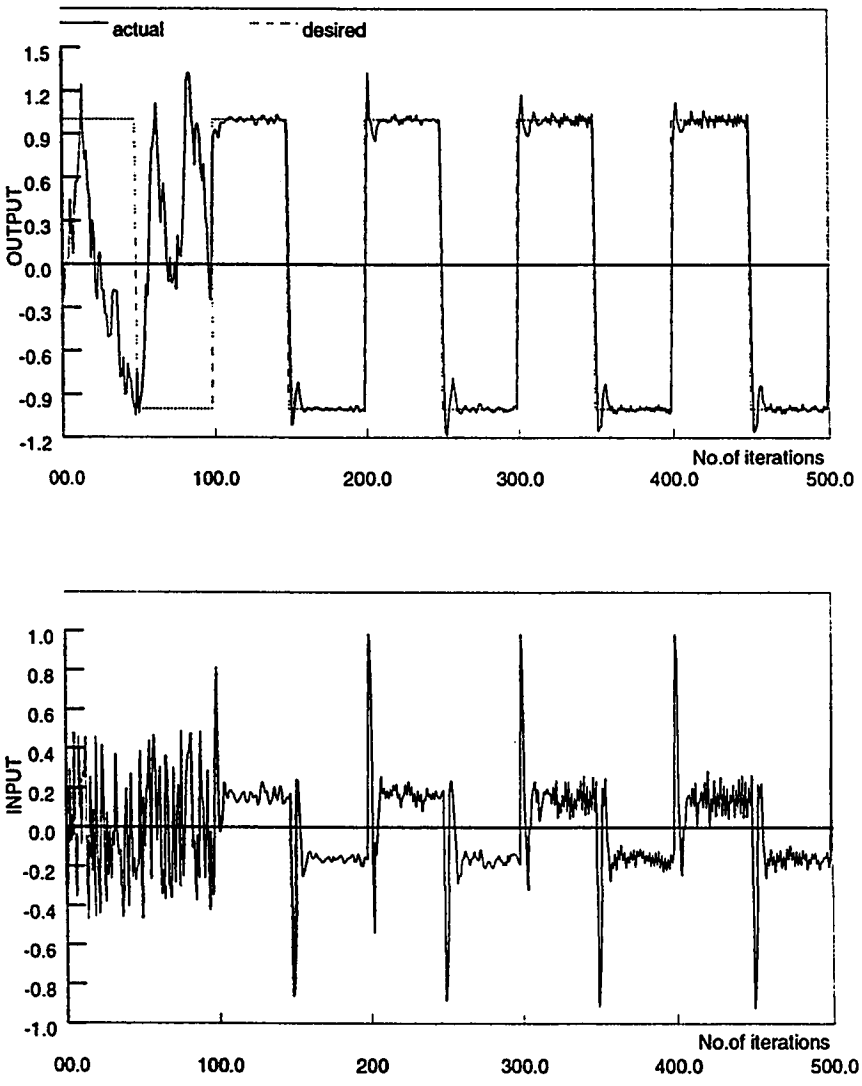


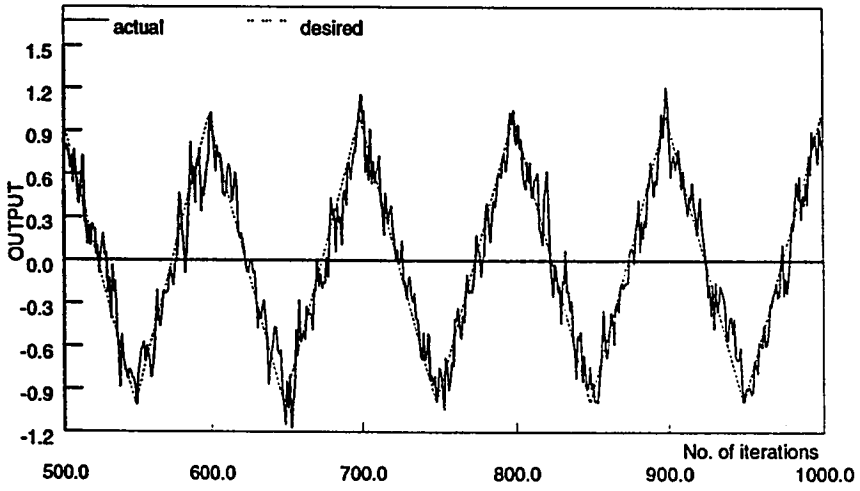
Figure 5.8: Minimum variance control of the stochastic Hammerstein model without estimating the noise terms-step set point

It has been indicated earlier, that in the linear case estimation of noise terms and noise parameters are not necessary. When the setpoint is piecewise constant, a nonlinear plant may well be approximated by piecewise linear models. It may be argued that in such a situation, inclusion of the noise terms in the neural net input may be avoided without severely affecting the performance. Figure 5.8 depicts such simulation, where the noise terms were not fed to the neural net input. It can be observed that the control performance in both cases is quite similar with only the transients being affected.

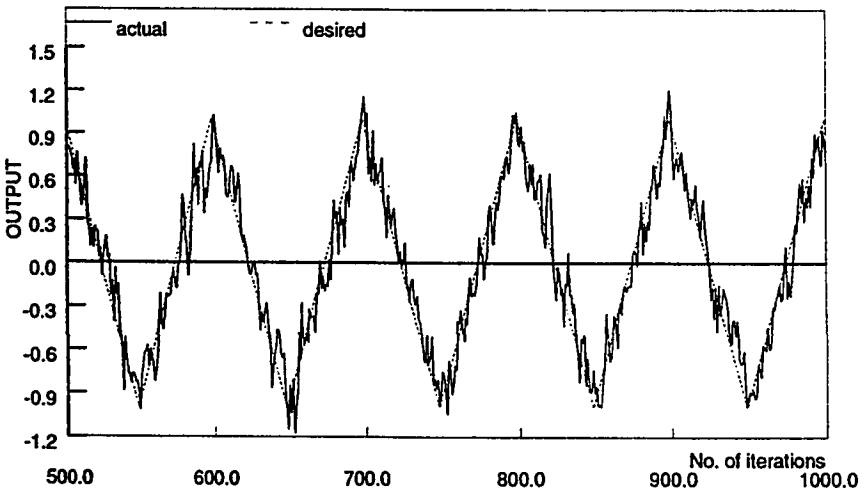
Figure 5.9 provides the simulation results when the setpoint is continuously varied. Even in this case it can be seen that the plant performance, both while including the noise terms in the net input and while excluding them is quite similar. This provides further justification for the recommendation that in order to reduce the computational complexity, feeding the noise terms $\hat{e}(k)$ to the neural net input may be avoided.

Figure 5.10 provides the simulation results on a Hammerstein model given by Plant IV. The dither has been used for the first 100 iterations while the set point has been switched between +1 and -1. The figure shows the plant performance with and without feeding the noise terms to the neural net input. As expected the plant performance differs only during the transient stage. The number of neurons in the hidden layer during both the simulations has been kept at 6. Simulation results on the same model with a sinusoidal setpoint are shown in figure 5.11, which leads to identical conclusions. The neural network learns the plant dynamics within about 100 iterations.

Figure 5.12 provides the simulation results using the Weiner model. The dither

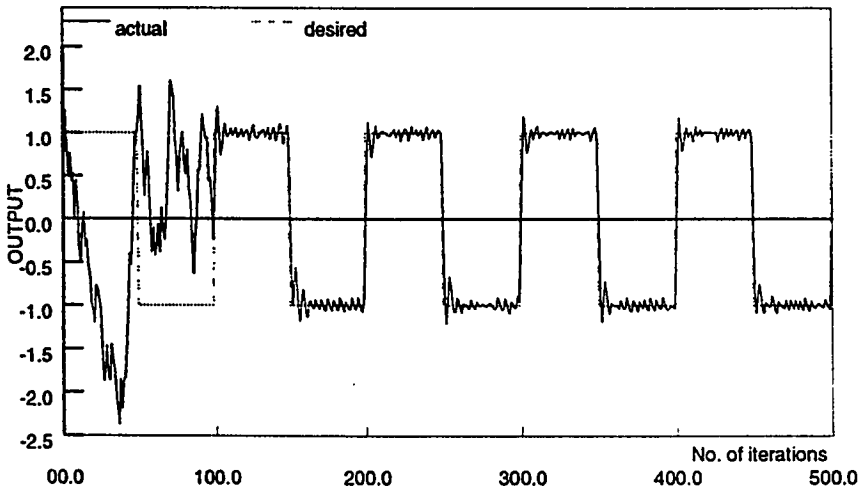


(a) Estimating the noise terms

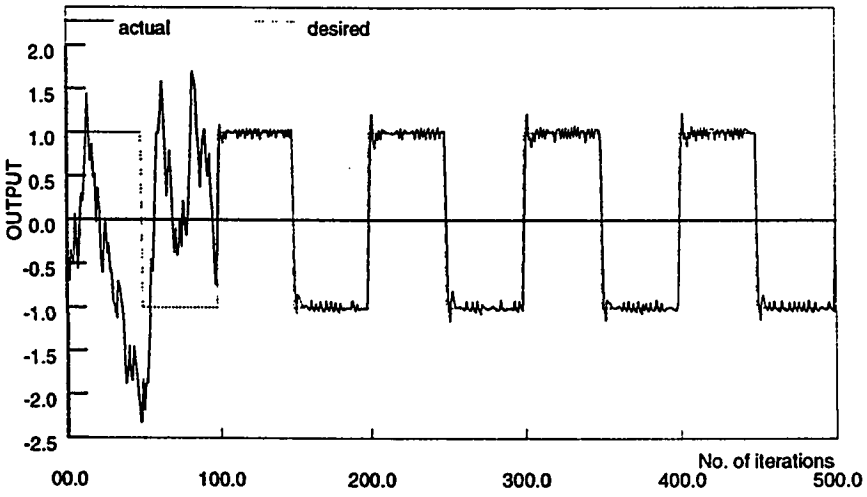


(b) Not estimating the noise terms

Figure 5.9: Minimum variance control of the stochastic Hammerstein model – triangular set point

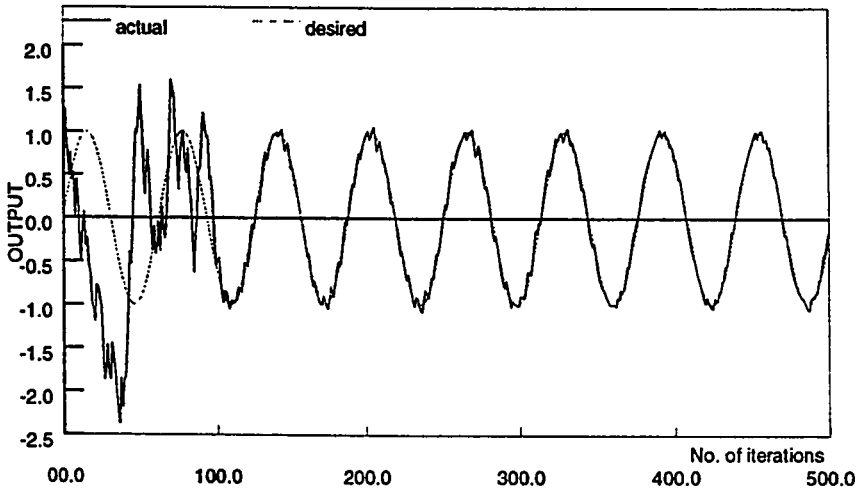


(a) Estimating the noise terms

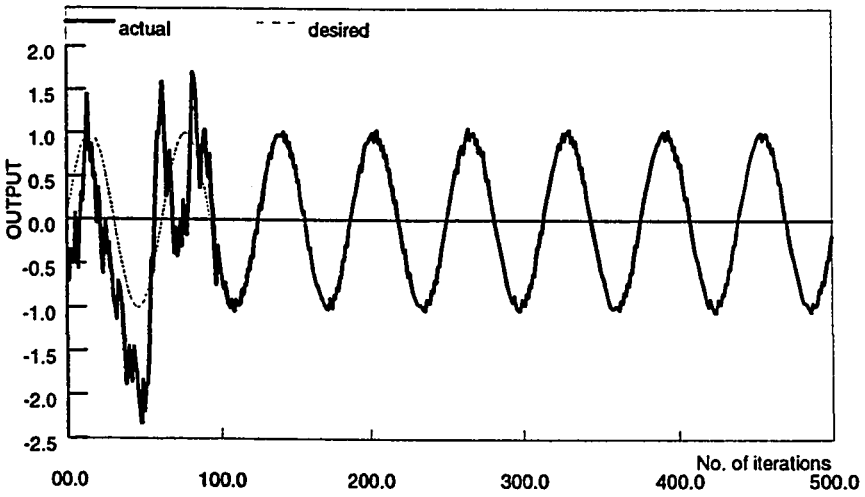


(b) Not estimating the noise terms

Figure 5.10: Minimum variance control of the stochastic Hammerstein model II



(a) Estimating the noise terms



(b) Not estimating the noise terms

Figure 5.11: Minimum variance control of the stochastic Hammerstein model II without estimating the noise terms-sinusoidal set point

has been used for the first 200 iterations while the set point has been switched between +5 and -5. The terms corresponding to noise is not being estimated in this case. However the tuning time for this example seems to be longer. A problem in the case of the nonlinear plants is that the true values of the weights of the neural network cannot be known and hence no information about the parameter convergence can be obtained.

The learning rates can be shown in terms of the sum squared error (SSE) between the actual and the estimated input. SSE using the minimum variance strategy have been shown in figure 5.13. Figure 5.13(a) shows the results of using the variable and constant learning rates for the NARX model operating in a stochastic environment with a noise σ of 0.01. The superiority of the variable learning rate is further borne out by the plots. Figure 5.13(b) shows the results on the Hammerstein model using the variable step size. The two learning rates shown therein correspond to the two cases of the neural net being with and without the noise terms as indicated by "estimating C" and "not estimating C" respectively. It can be observed that excluding the use of the noise terms in the neural network does make training time faster. This results from the reduction of the neural net size.

5.5 Generalized minimum variance strategy

5.5.1 Stochastic linear plants

The plant is considered to be described by

$$y(k) - y(k-1) + 0.24y(k-2) = u(k-1) + 1.5u(k-2) + \xi(k) + 0.5\xi(k-1) \quad (5.9)$$

A single layered neural network with linear activation units is also used. The

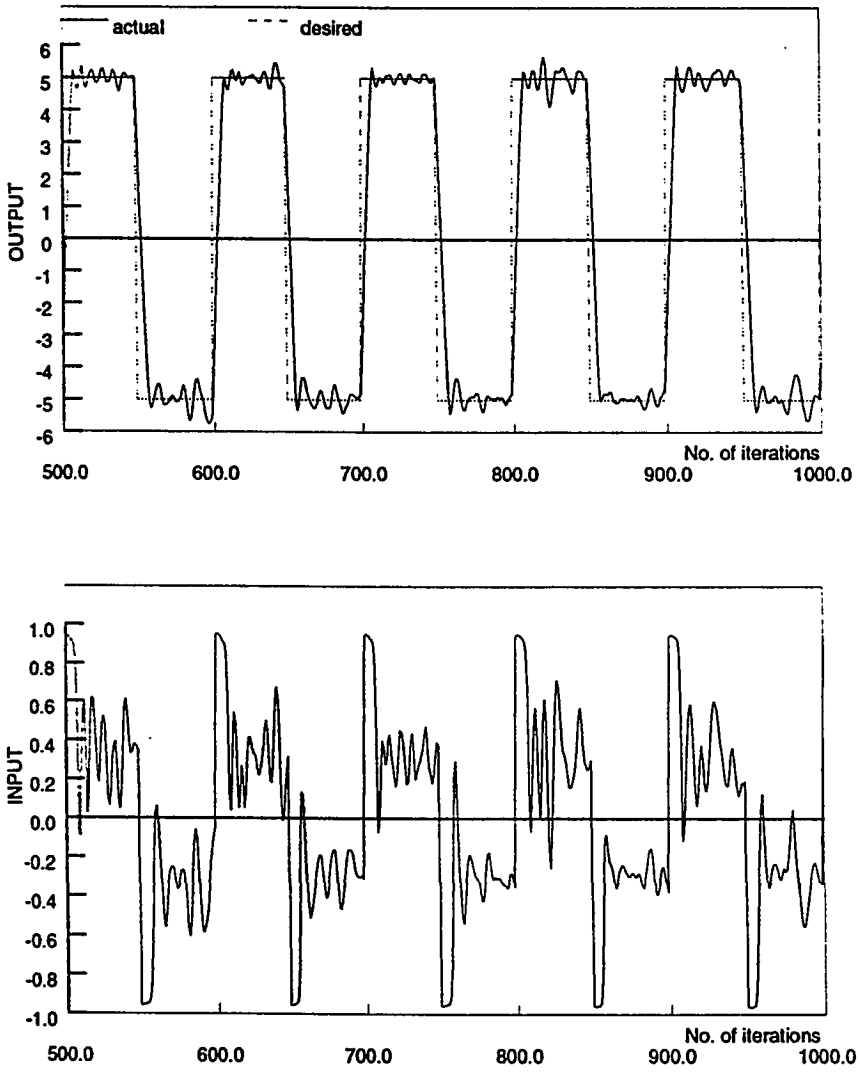


Figure 5.12: Minimum variance control of the stochastic Weiner model without estimating the noise terms-step set point

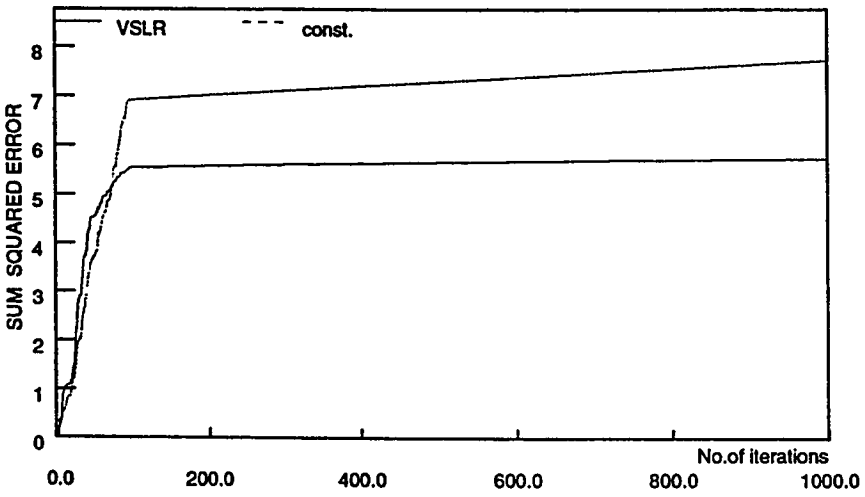
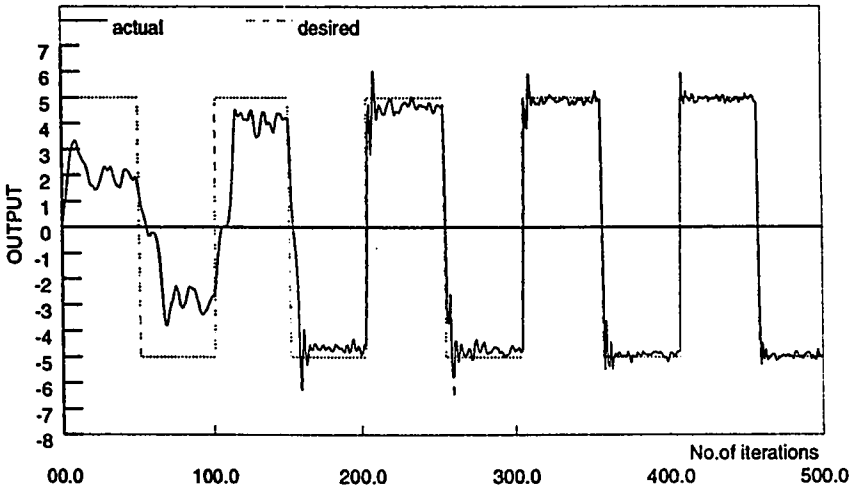


Figure 5.13: Learning rates using the minimum variance strategy

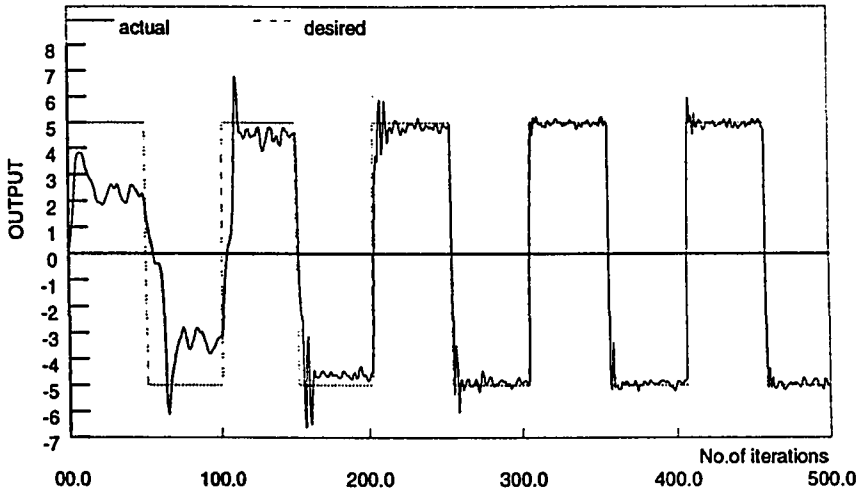
desired set point was varied between +5 and -5 after every 50 iterations. The value of λ used has been 1. A random normally distributed noise with a σ of 0.1 is assumed. μ_0 has been taken as 1.98 while $s(0)$ is 1.0. The simulation results from using equation 5.2 and strategy I are shown in figure 5.14. It can be seen that the neural network has learnt the plant characteristics in about 400 iterations. Further, the overshoot at switchover of the setpoint is also neglible after 400 iterations. Initially, though it was significant in the region about 300 iterations.

Figure 5.14(b) also shows the plant performance when the coefficients corresponding to the noise terms are not estimated. As can be seen from figure 5.14, results obtained when the noise parameters aren't estimated are quite similar to those obtained with estimation of the noise parameters. The difference though is that the transients have been affected as expected. It is also observed that in all of the above cases, adding dither to the input for the first few iterations does accelerate the learning process.

The approach of generalized minimum variance can also be used to control minimum phase plants. It is observed that this overcomes the problem of ringing in the control input (due to the bad location of a zero). This is shown in figures 5.15 and 5.16. The zero of the plant in this case is at -0.5 while the poles are at -0.9 and -0.7. As shown in figure 5.15, with a minimum variance controller ringing is present in the input. This is removed when a generalized minimum variance controller is used to control the plant. Figure 5.16 depicts this fact. The weighting on the control input in this case has been set at 5. However, the overshoot in the output has increased. Thus the control costing parameter λ should be a trade off value so as to have a compromise between the smooth input and an acceptable output. The values of the



(a) Estimating the noise terms



(b) Not estimating the noise terms

Figure 5.14: Stochastic linear plant using the GMV strategy I

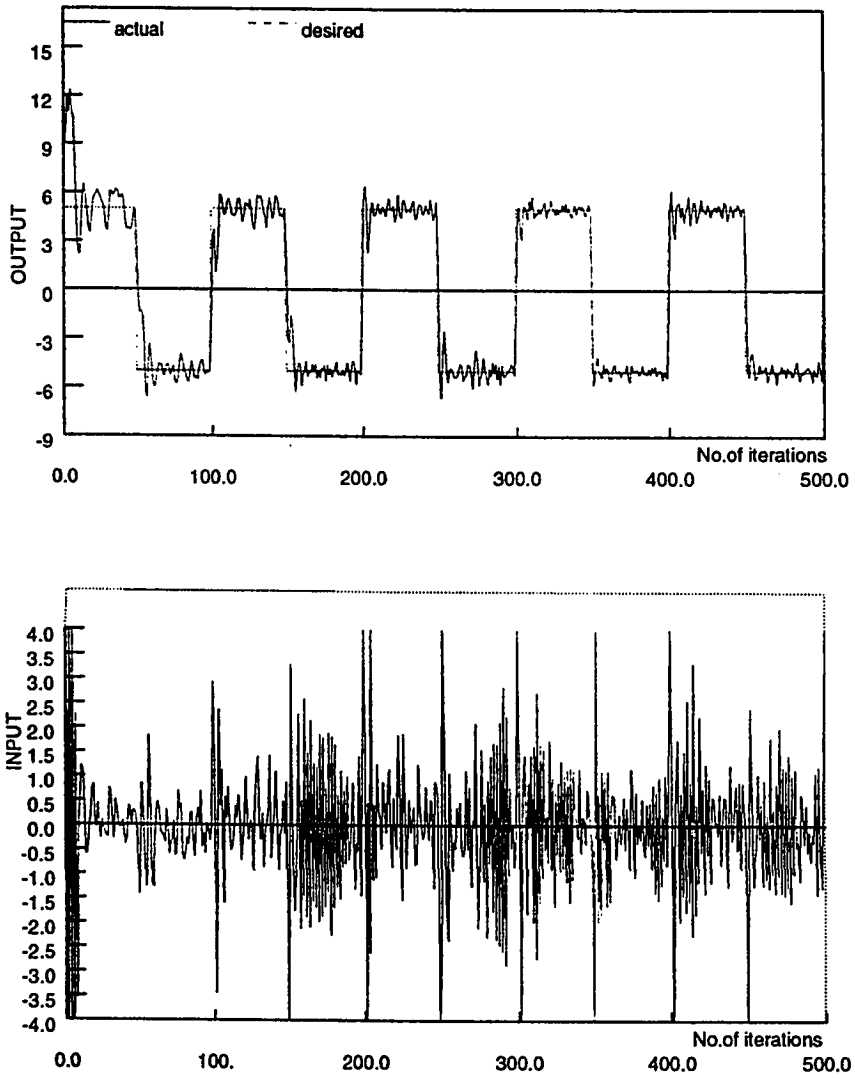


Figure 5.15: Ringing in input using MV strategy

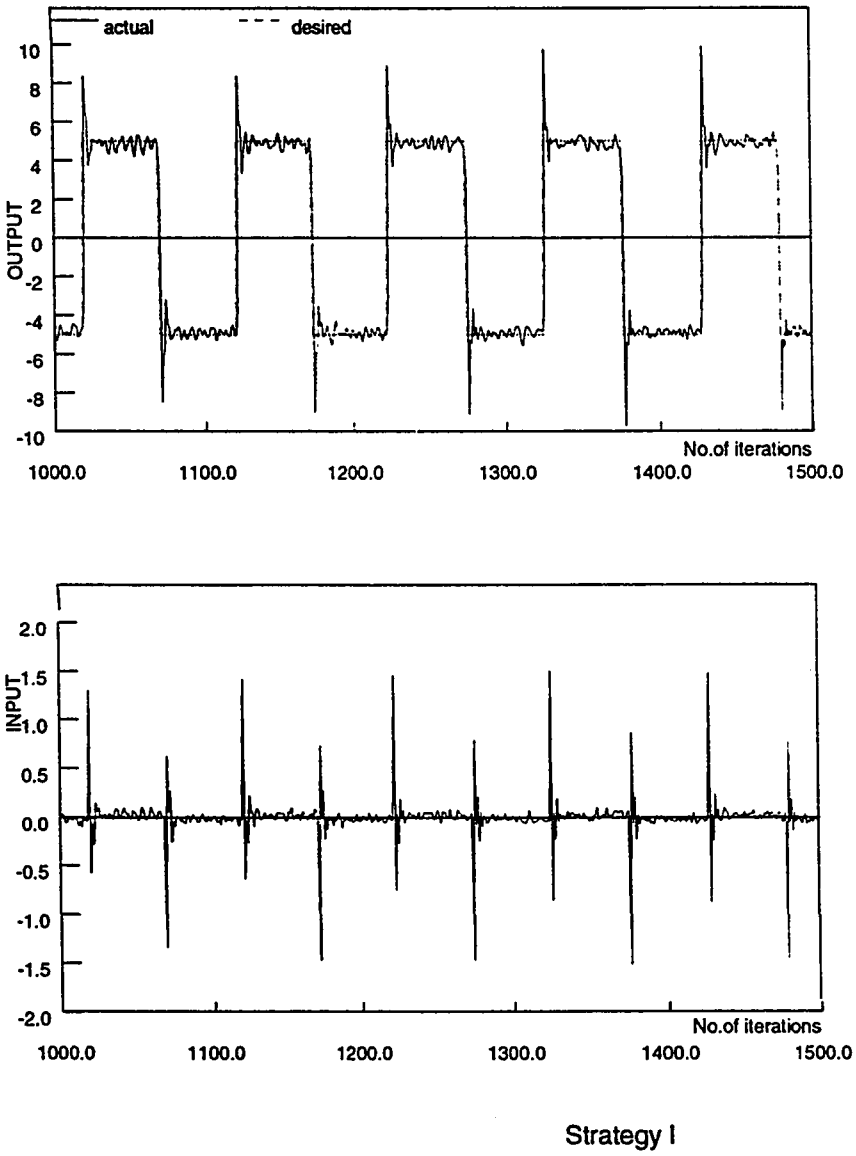


Figure 5.16: Absence of ringing using GMV strategy I

other parameters are as in the earlier examples.

5.5.2 Stochastic nonlinear plants

The plant is assumed to be represented by

$$y(k)-1.6y(k-1)+0.63y(k-2) = u'(k-1)+1.5u'(k-2)+\xi(k)+0.5\xi(k-1) \quad (5.10)$$

with

$$u'(k) = \begin{cases} 0 & \text{if } \mid u(k) \mid \leq 0.1 \\ u(k) - 0.1 \text{ sign } \{u(k)\} & \text{otherwise} \end{cases}$$

The linear part of the plant has a zero at -1.5. When the minimum variance strategy is applied to this plant, as expected the algorithm blew up. Next, the generalized minimum variance strategy has been applied. ρ is chosen as 0.25. The $\hat{h}$ required in the computation of $\phi(k)$(see 4.57) is computed using back propagation. However, in order to avoid division by small quantity during the initial phase of the training, the expression for $\phi(k)$ is modified as

$$\phi(k) \;=\; y(k) + \frac{\rho h}{h^2(k)+\tau} u(k-d) \qquad (5.11)$$

where τ is a small positive quantity which has been chosen as 0.001. Figures (5.17) and (5.18) respectively show the simulation results using the strategies I and II as outlined in the previous chapter. In both cases, dither has been added to the setpoint for the first 1000 iterations. As expected the use of strategy I failed to eliminate the offset. To overcome the problem the second strategy was used. The simulation results are shown in figure 5.18. It can be seen that the offset is overcome using this strategy. The input demanded by the plant are also shown in the figures. Further reductions in the input magnitudes or variations can be obtained by increasing the control costing parameter ρ.

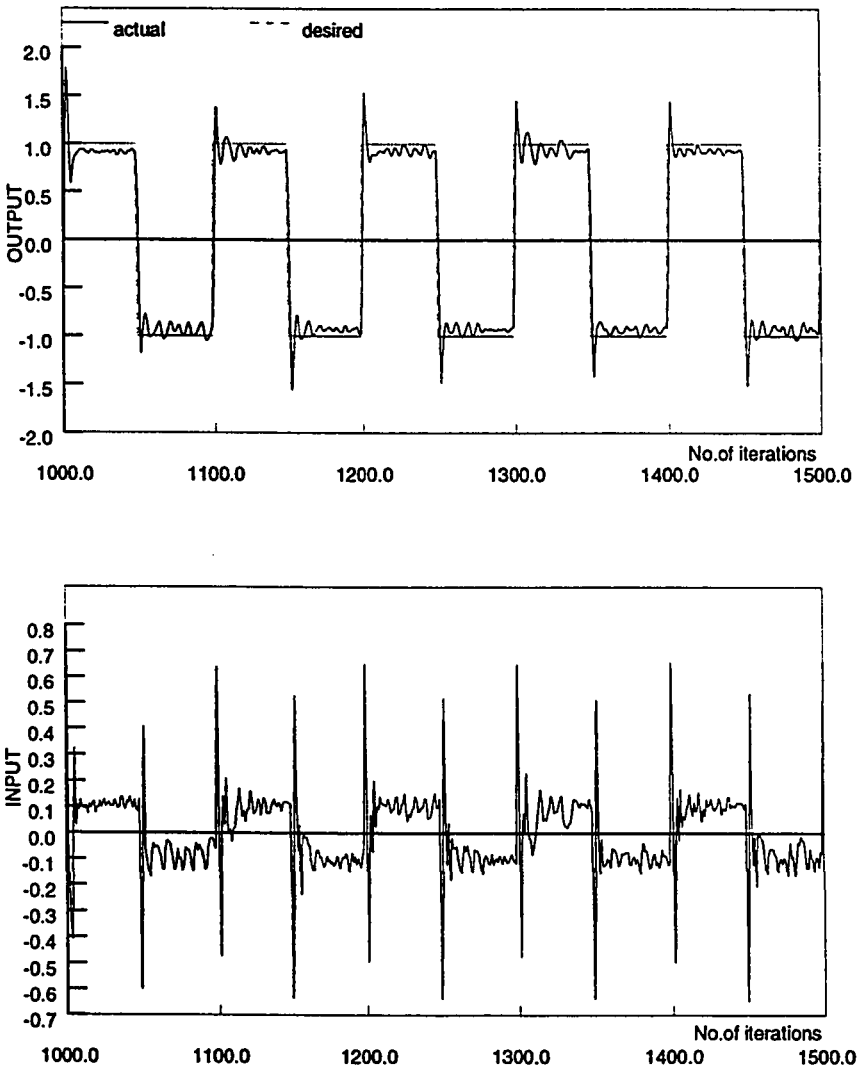


Figure 5.17: GMV control of an inverse unstable plant with feedforward adaptation

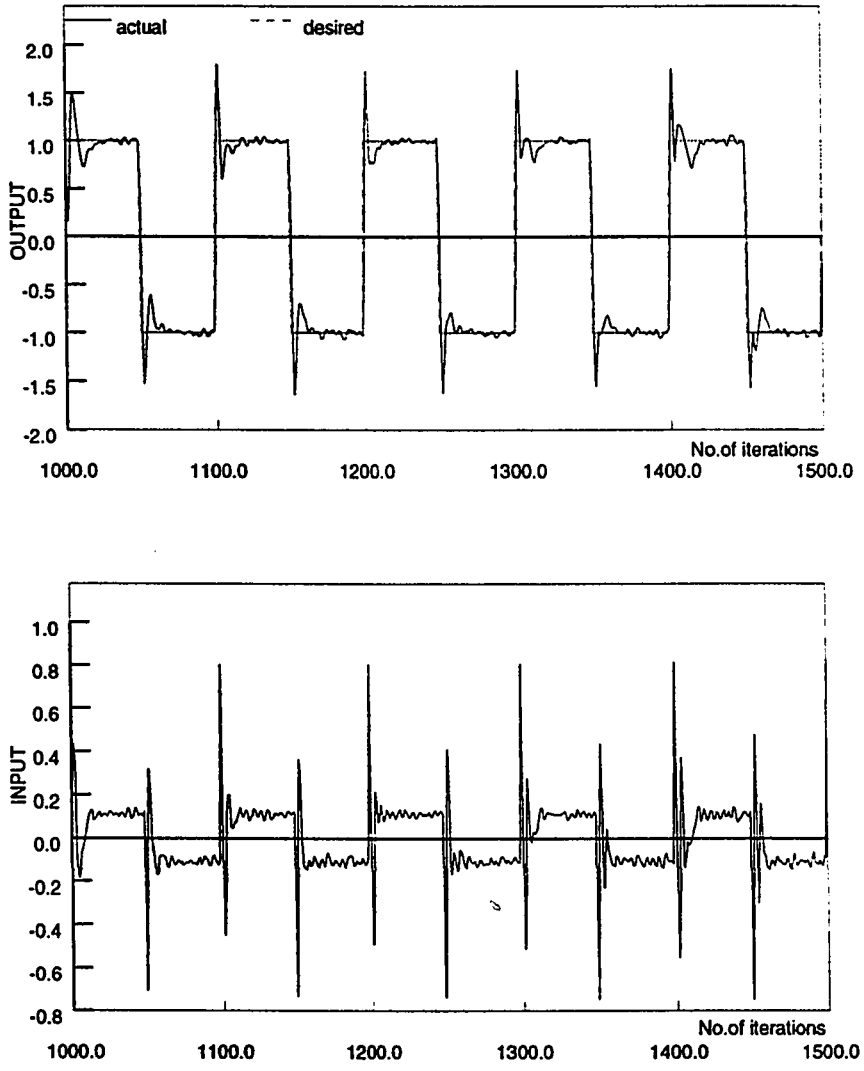


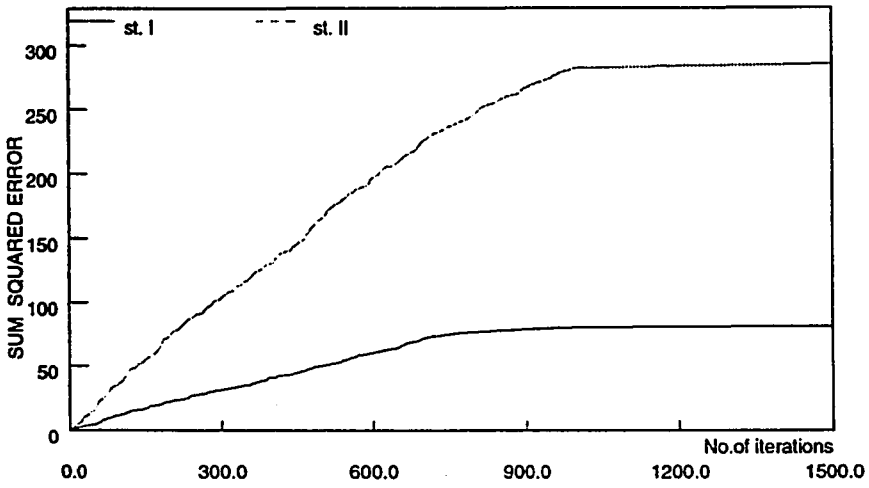
Figure 5.18: GMV control of an inverse unstable plant using an integrator

The learning rates using the generalized minimum variance strategy have been shown in figure 5.19 in terms of the SSE. Figure 5.19(a) shows the results using strategies I and II in the deterministic case on the inverse unstable Hammerstein model. It is also seen that the learning performance of strategy I is better compared to that of strategy II. This is expected since in the case of strategy II the neural network has more parameters. Further, while the ultimate flattening of the curve is expected only in the deterministic case, it appears that it occurred even in the stochastic case. This is due to the scale and low value of σ(=0.01).

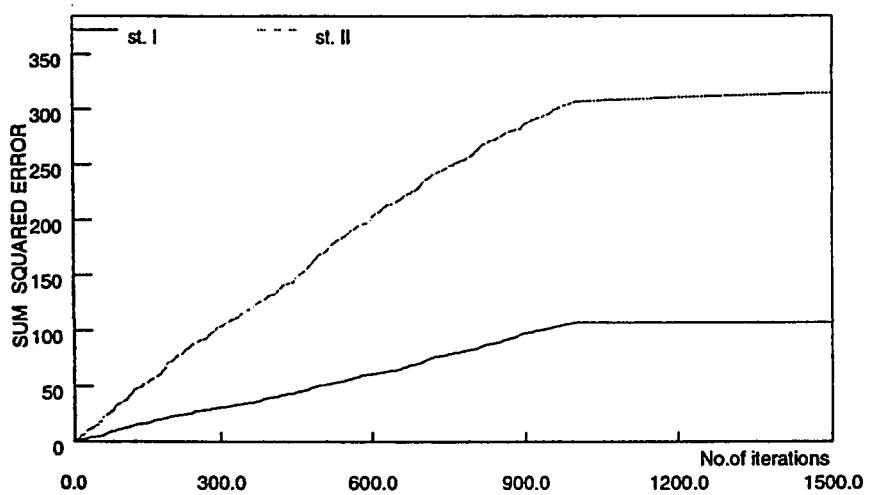
A model identified from a practical system which is a laboratory scale liquid level plant is given in [42]. The system is a second-order one and consists of a DC water pump feeding a conical flask which in turn feeds a square tank. The controllable input is the voltage to the pump motor and the system output is the height of the water in the conical flask. The simulated control objective is for the water height to follow some demand signal. The plant model has been identified as

$$\begin{aligned} y(k) \;=\; & 0.9722y(k-1)+0.3578u(k-1)-0.1295u(k-2)-0.3103y(k-1)u(k-1) \\ & -0.04228y^2(k-2)+0.1663y(k-2)u(k-2)-0.03259y^2(k-1)y(k-2)- \\ & 0.3513y^2(k-1)u(k-2)+0.3084y(k-1)y(k-2)u(k-2)+ \\ & 0.1087y(k-2)u(k-1)u(k-2)+0.2573y(k-2)e(k-1) \\ & +0.2939y^2(k-2)e(k-1)+0.4770y(k-2)u(k-1)e(k-1) \end{aligned} \tag{5.12}$$

The set point has been varied between 0.25 and -0.25 after every 50 iterations.The results of the GMV control of the plant using strategy II are shown in figure (5.20), with ρ chosen as 0.2. It can be seen that the neural network has learnt the plant characteristics reasonably well within about 400 iterations.



(a) Deterministic case



(b) Stochastic case

Figure 5.19: Learning rates using the generalized minimum variance strategy

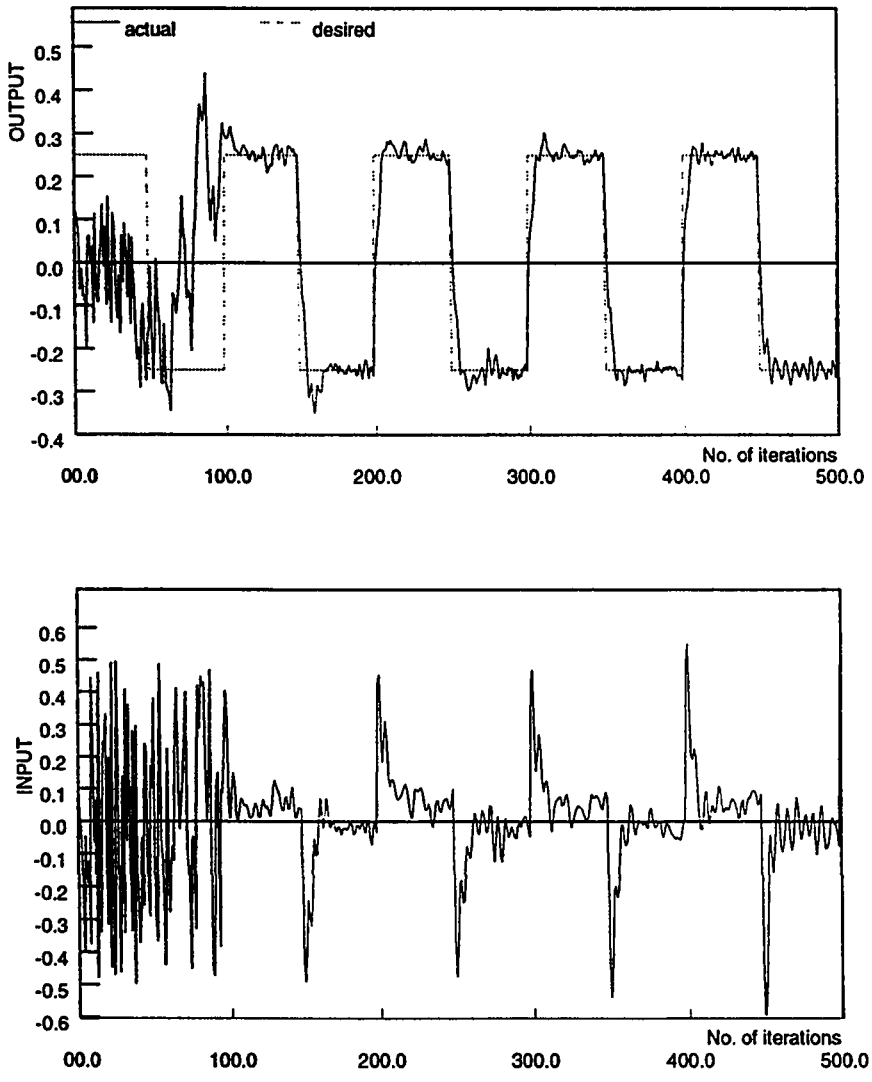


Figure 5.20: GMV control of the liquid level of a simulated second order plant

Chapter 6

CONTRIBUTIONS, CONCLUSIONS AND RECOMMENDATIONS

An area of control systems which offers much scope for the use of neural networks is adaptive control. Here the controller has to adapt itself with a change in the process parameters. A class of adaptive control algorithms is called self-tuning. A self-tuning algorithm is one in which the control signal generated from estimated parameters turns out to be asymptotically the same as the control law generated by feedback, if the process parameters were known. In this work, we have concentrated upon methods of utilizing neural networks as implicit self-tuning controllers for both linear as well as nonlinear plants. Inverse stable as well as inverse unstable plants have been considered.

6.1 Contributions

The main contribution in this thesis can be summarized as follows.

1. The minimum variance self tuning regulator of Astrom [4] has been extended to nonlinear plants.

2. The generalized minimum variance self tuning algorithms proposed by Clarke and Gawthrop [7] have also been extended to nonlinear systems. This extension is capable of controlling unstable plants.

3. Self tuning algorithms for both deterministic as well as stochastic nonlinear plants have been treated.

4. The use of variable learning rate capable of improving the convergence of a self tuning algorithm has been proposed and implemented.

5. All the proposed extensions have been analyzed theoretically in terms of linear systems.

6. Extensive simulation studies have been carried out to establish the feasibility of the propositions as well as to validate the theoretical developments.

7. Different heuristics have also been proposed in order to overcome problems associated with the implementation of the proposed controllers. These include the choice of μ for the variable learning rate, methods to prevent the neural network from locking, thereby not learning at all and also means of preventing the algorithm instability due to poor parameter estimation.

6.2 Conclusions

1. Neural networks can be effectively used in the control of linear as well as nonlinear plants as direct or implicit self tuning controllers.

2. Similar to linear plants, direct self tuning control algorithms can also be effectively used to control inverse stable as well as inverse unstable nonlinear plants.

3. Neural net based direct self-tuning algorithms can also be designed to accomodate stochastic nonlinear plants.

4. The convergence of neural net learning can be improved through use of variable learning rate. The variable learning rate schemes used in our study is based on linearization of the error in the weight space.

5. Further it has also been seen through simulation study that, in the case of noisy plants the estimation of noise terms may be bypassed. This may reduce the computational requirements.

6.3 Recommendations

1. Consequent upon the easy availability of high power computing at low cost practical implementations of the proposed algorithms can be undertaken.

2. Extensions to MIMO case can also be considered.

3. Theoretical study can be undertaken to analyse the proposed schemes in order to gain insight about local and global stability properties.

4. Comparisons can also be undertaken with other schemes using neural networks, such as neural network controllers based on the MRAC strategy etc.

5. Schemes to use neural networks as controllers based on off-line design strategies can also be investigated.

Appendix A

A.1 Minimum variance strategy

A.1.1 Deterministic linear plants

Consider a linear system described in the prediction form by

$$y(k) + A(z^{-1})y(k-d) = B(z^{-1})u(k-d) \tag{A.1}$$

where u is the control variable and y is the output. The delay is denoted by d. where

$$A(z^{-1}) = a_0 + \cdots + a_{n_a} z^{-n_a} \tag{A.2}$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} + \cdots + b_{n_b} z^{-n_b} \tag{A.3}$$

$$b_0 \neq 0$$

The plant description can be rewritten as an inverse dynamics model as

$$u_a(k-d) = \frac{1}{b_0} y(k) + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u_a(k-d) \tag{A.4}$$

where

$$\bar{A}(z^{-1}) = A(z^{-1})/b_0 = (a_0/b_0) + \cdots + (a_{n_a}/b_0)z^{-n_a} \tag{A.5}$$

$$\bar{B}(z^{-1}) = -\frac{B(z^{-1})}{b_0} + 1 \tag{A.6}$$

$$= -(b_1/b_0)z^{-1} - (b_2/b_0)z^{-2} - \cdots - (b_{n_b}/b_0)z^{-n_b} \tag{A.7}$$

and y(k) is the output at time k .

Hence an estimate of u(k-d) can be written as

$$\hat{u}(k-d) = \frac{1}{b_0}y(k) + \bar{A}(z^{-1})y(k-d) + \bar{B}(z^{-1})u(k-d) \tag{A.8}$$

where use of certainty equivalence principle is being made use of. The estimated input can also be expressed as

$$\hat{u}(k-d) = (1/b_0)y(k) + \sum_{i=0}^{n_a}\alpha_i y(k-d-i) - \sum_{i=1}^{n_b}\beta_i u(k-d-i) \tag{A.9}$$

where $\alpha_i = a_i/b_0$ and $\beta_i = b_i/b_0$.

Neural Network Controller

Now assuming that the parameters α_i and β_i have been estimated correctly we can have a dead beat controller as

$$u(k-d) = (1/b_0)y_d + \sum_{i=0}^{n_a}\alpha_i y(k-d-i) - \sum_{i=1}^{n_b}\beta_i u(k-d-i) \tag{A.10}$$

When the parameters α_i and β_i are properly identified then $y(k)$ becomes y_d and

$$u(k-d) - \hat{u}(k-d) = e(k) = 0.$$

Hence the output of the plant tracks the set point as desired. When a single layered neural network with linear activation units is used as a controller the desired values of the weights are the α_i and the β_i given above. α_i and β_i are estimated by minimizing

$$J = \frac{1}{2}\|e(k)\|^2$$

A.1.2 Deterministic nonlinear plants

Assume a nonlinear model to be of the form

$$y(k) = f[y(k-d), y(k-d-1), \cdots, \cdots, y(k-d-n_y); u(k-d), u(k-d-1), \cdots . u(k-d-n_u)] \quad \text{(A.11)}$$

Hence the estimate of the control input is given by

$$\hat{u}(k-d) = g[y(k), y(k-d), \cdots . y(k-d-n_y); u(k-d-1), \cdots . u(k-d-n_u)] \quad \text{(A.12)}$$

where g represents an inverse function.

Neural network controller

Following the approach of the deterministic linear systems the output of the neural network controller is given as

$$u(k-d) = g[y_d, y(k-d), \cdots, y(k-d-n_y); u(k-d-1), \cdots . u(k-d-n_u)] \quad \text{(A.13)}$$

As in the case of the linear system the identification error is defined as

$$e(k) = u(k-d) - \hat{u}(k-d) \quad \text{(A.14)}$$

If the nonlinear function g, which is the inverse model of the plant, has been properly identified, then in the absence of any plant noise the error e(k) would become zero. Thus the weights of the neural network could be adjusted by minimizing $||e(k)||^2$.

A.2 Simulation results

Deterministic linear plant

The plant is assumed to be described by the following difference equation

$$y(k) - 1.6y(k-1) + 0.63y(k-2) = u(k-1) - 0.5u(k-2) \qquad \text{(A.15)}$$

The poles of the plant are located at 0.9 and 0.7 while the zero is at 0.5. Thus the plant is minimum phase. Since the plant is linear, hence for the purpose of simulation, only a single layer neural network is used. Further, the activation function for the different neurons in the neural network is assumed to be unity.

For set-point tracking, the reference $y_d(t)$ is varied between ± 5. The setpoint was changed every 50 iterations. The control strategy used is the minimum variance one. A constant step size as well as a variable scalar learning rate are used in the backpropagation algorithm. The constant step size has been 0.01 while the variable scalar learning rate used is given by

$$\eta = \frac{\mu}{\epsilon + ||\Gamma||^2} \qquad \text{(A.16)}$$

where Γ is the gradient of the neural network obtained through backpropagation. ϵ is a small quantity which is taken as 0.001. It is used for the purpose of avoiding singularity problems.

The control results obtained from the simulation using the variable and constant scalar learning rates respectively are shown in figures A.1 and A.2. The constant scalar learning rate has been taken as 0.01. In the case of the variable scalar learning rate, the values of the different parameters are $\mu_\infty = 1$, $\mu_0 = 1.7$ and $\mu_r = 0.97$.

It can be seen that satisfactory set point following is achieved. It is also observed that the use of the variable scalar learning rate results in an improvement in the training rate of the neural network. The parameter estimation error given by the Euclidean norm, which is a measure of how close the estimated weights are to the true weights, is greatly reduced when the variable scalar learning rate is used. A plot of the norm is shown in figure A.1 and figure A.2 for the case of variable step size and constant step size respectively. It has also been observed that by using a random input for about 50 iterations, the convergence rate can be improved as evidenced from different measures such as the norm, set point following etc. This is mainly because dither helps in exciting the different frequencies of the system, thus facilitating rapid identification of the plant.

A.2.1 Nonlinear deterministic plants

The plants simulated have been a Hammerstein model and a Weiner model shown in figure A.3.

Since the plants are nonlinear, a two layered neural network has been used. The nonlinear activation unit is taken as the hyperbolic tangent. One biased neuron is used in each of the input and the hidden layers. Six neurons were taken in the hidden layer. The parameters of the neural network are adjusted using the static backpropagation method and the variable scalar learning rate as given in equation A.16.

For the Hammerstein model the desired setpoint is switched between +5 and -5. A dither signal is added to the input for the first 200 iterations. The simulation results obtained using the variable step size are shown in figure A.4. Using the

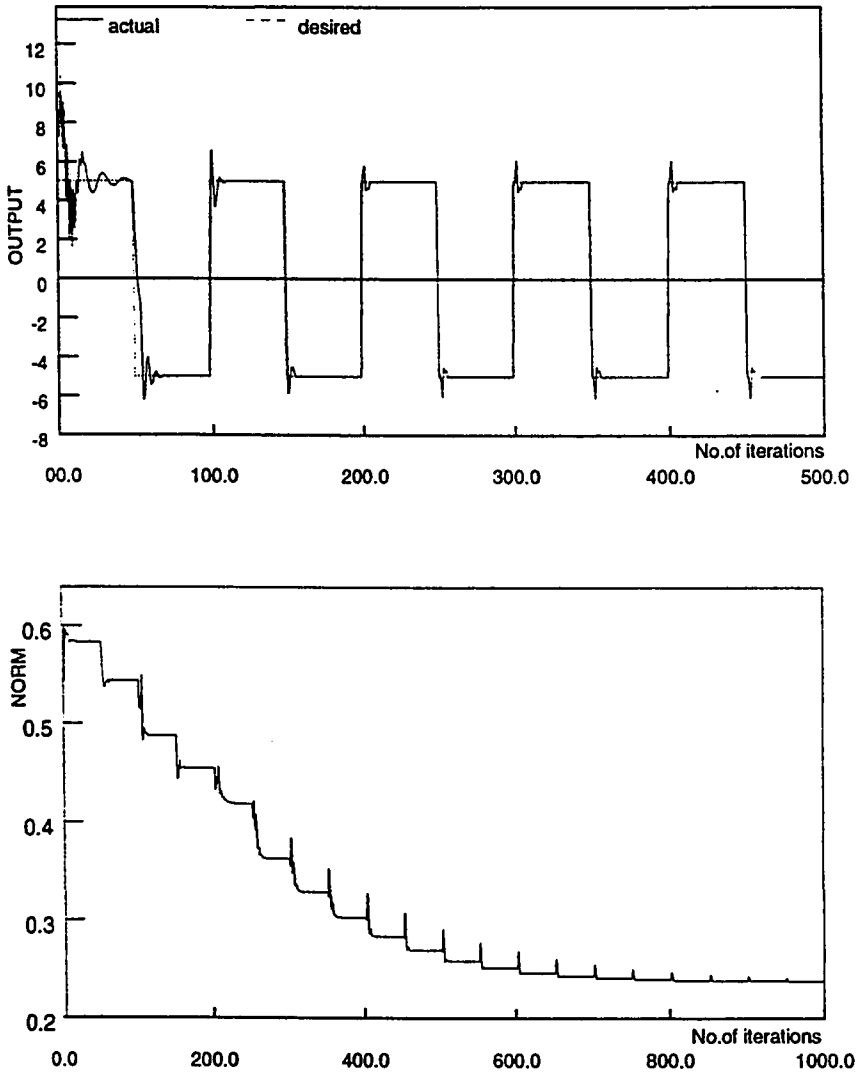


Figure A.1: Deterministic linear plant using minimum variance strategy and variable step size

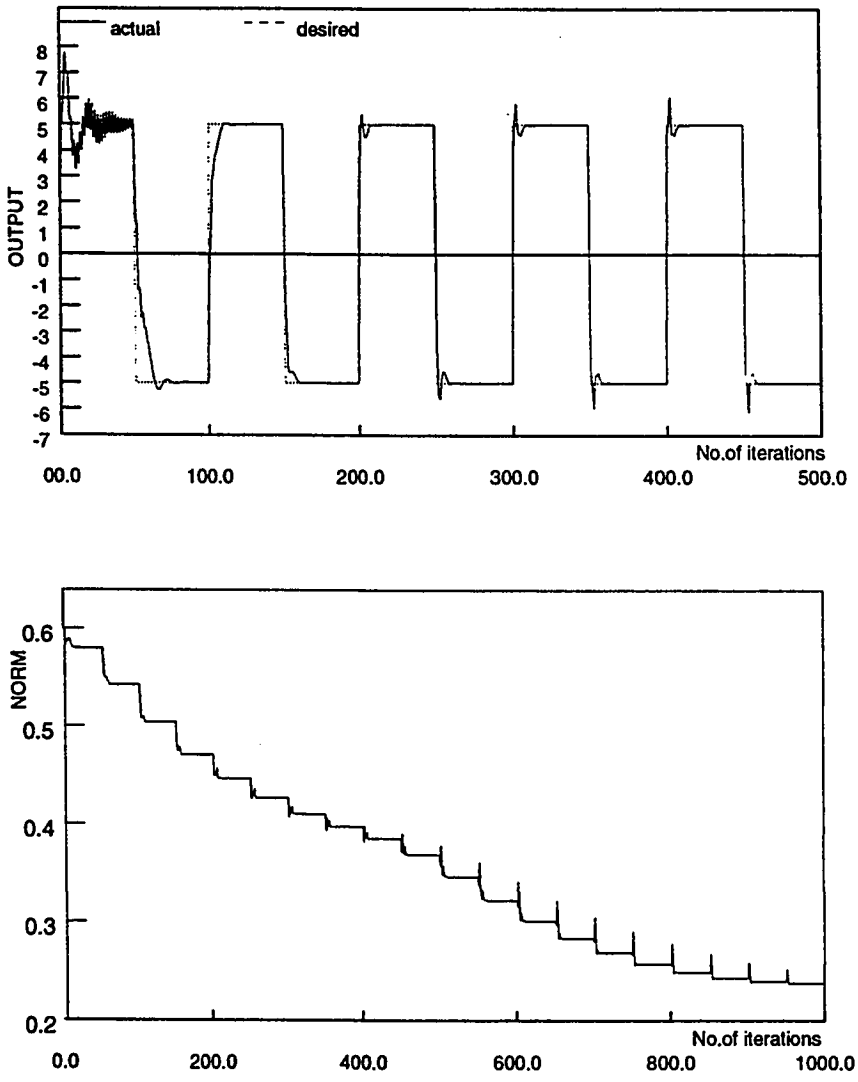
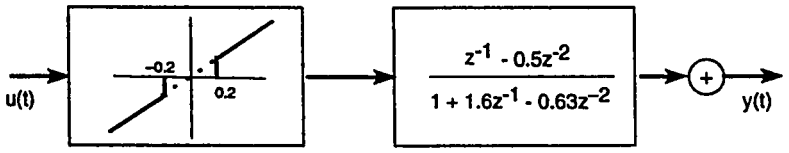
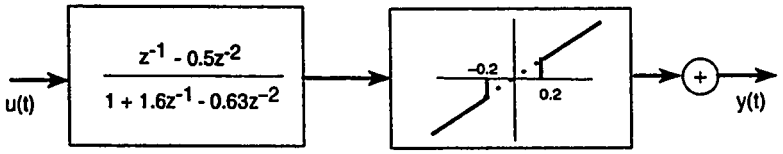


Figure A.2: Deterministic linear plant using minimum variance strategy and constant step size



Plant II: The Simulated Hammerstein Model



Plant III:The Simulated Weiner

Figure A.3: Nonlinear plant models

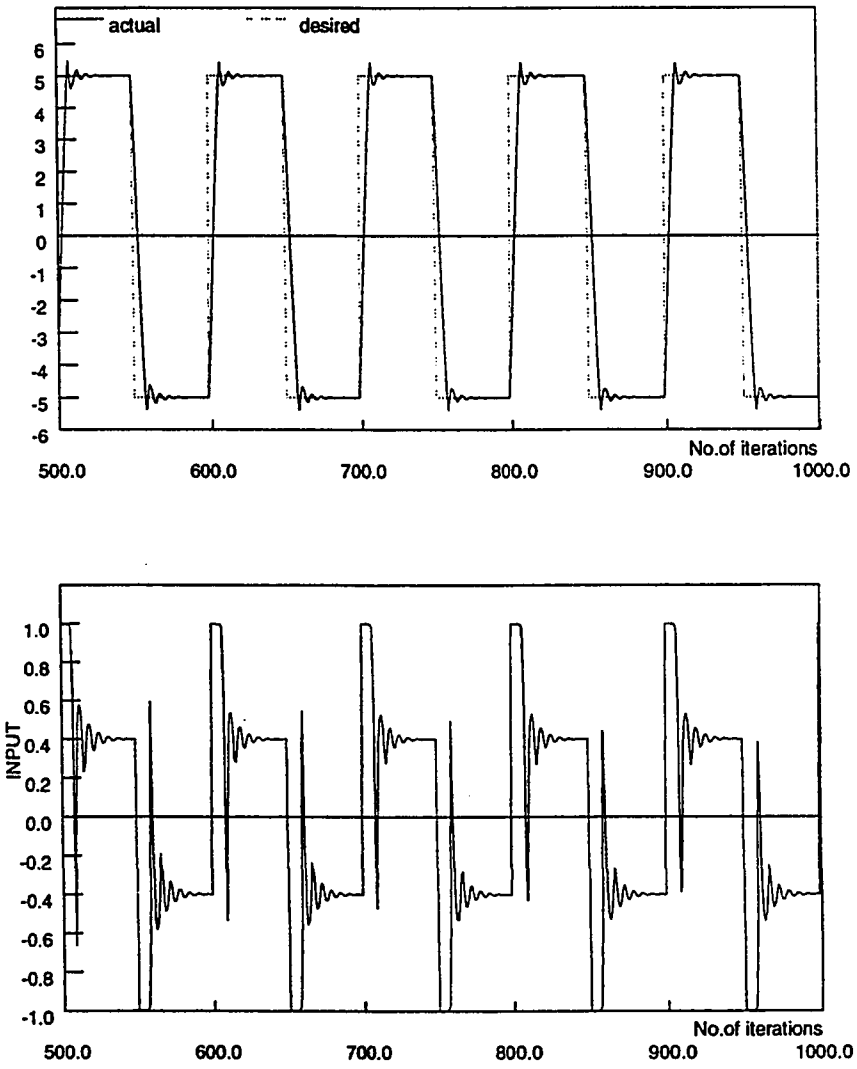


Figure A.4: Deterministic Hammerstein model using the MV strategy

variable step size in the backpropagation algorithm the neural network identifies the plant characteristics very quickly. As can be seen from the figure the actual plant output came in close conformity with the reference input within about 500 iterations. Yet, it is seen that with the rapid switching of the set point, control takes some time to track the changed setpoint. This is due to the fact that the control limits have been fixed at ± 1. The problem can be avoided if no limits on the input are imposed.

For the Weiner model also, the set point is switched between +5 and -5. A dither signal is applied to the plant for the first 200 iterations. After 200 iterations the control is switched on. It can be seen from figure A.5 that the neural network learns the plant dynamics fast enough. As a result the actual plant output follows closely the command signal to the system.

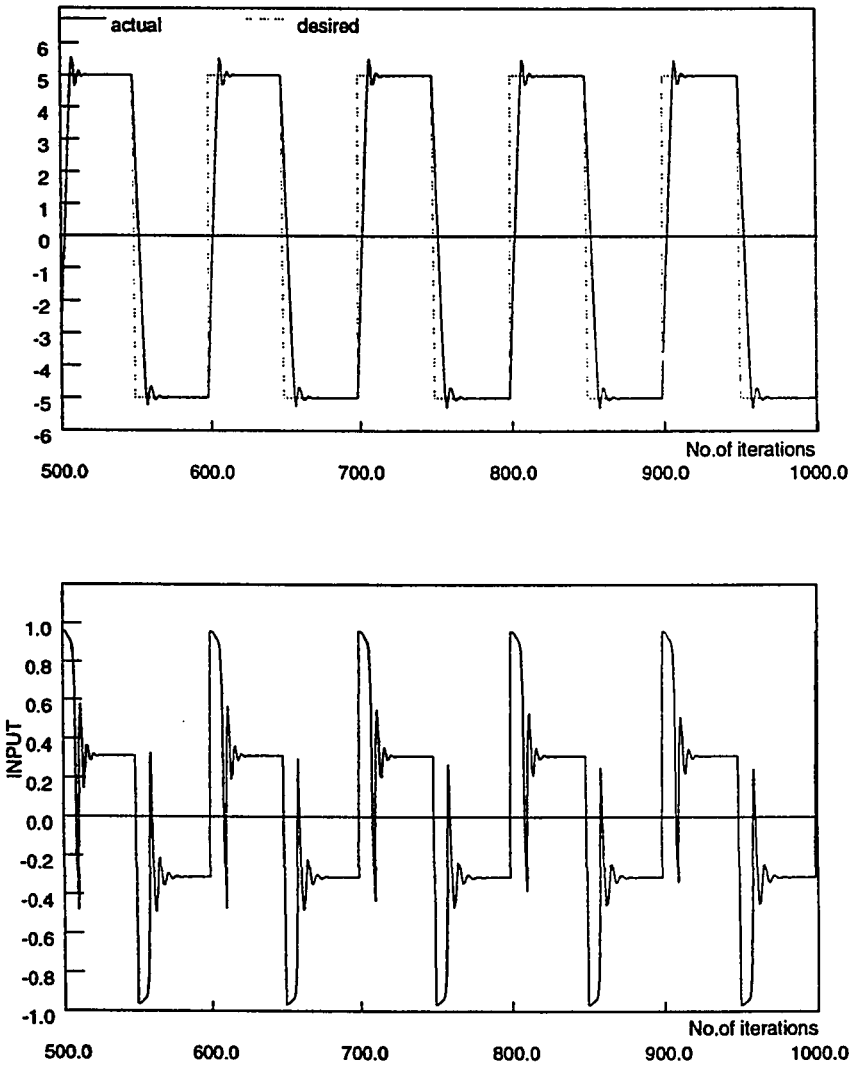


Figure A.5: Deterministic Weiner model using the MV strategy

Appendix B

B.1 Generalized minimum variance strategy

B.1.1 Linear deterministic plants

Let the plant be described as

$$\ddot{A}(z^{-1})y(k) = B(z^{-1})u(k-d) \tag{B.1}$$

$$\ddot{A}(z^{-1}) = 1 + z^{-d}A(z) = 1 + a_0 z^{-d} + \cdots + a_{n_a} z^{-d-n_a} \tag{B.2}$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} + \cdots + b_{n_b} z^{-n_b} \tag{B.3}$$

$$b_0 \neq 0$$

where one or more roots of B may be outside the unit circle. The pseudo output is given by

$$\phi(k) = y(k) + \lambda u(k-d) \tag{B.4}$$

From equations B.1 and B.4 we have

$$\ddot{A}(z^{-1})\phi(k) = [B(z^{-1}) + \lambda\ddot{A}(z^{-1})]u(k-d) \tag{B.5}$$

$$\text{or } \ddot{A}(z^{-1})\phi(k) = \tilde{B}(z^{-1})u(k-d) \tag{B.6}$$

123

where

$$\tilde{B} = [B(z^{-1}) + \lambda\ddot{A}(z^{-1})] \tag{B.7}$$

Hence the estimate of the control input to the plant is given by

$$\hat{u}(k-d) = \frac{1}{\tilde{b}_0}[\ddot{A}(z^{-1})\phi(k) - \{\tilde{B}(z^{-1}) - b_0\}u(k-d)] \tag{B.8}$$

where $\tilde{b}_0 = b_0 + \lambda$.

Neural Network Controller

Using strategy I of section 4.4, the controller attempts to minimize the variance of $\phi(k)$ by setting

$$u(k-d) = \frac{1}{\tilde{b}_0}[\ddot{A}(z^{-1})\phi(k) - \{\tilde{B}(z^{-1}) - b_0\}u(k-d)] \tag{B.9}$$

$$= \frac{\tilde{b}_0}{\tilde{b}_0 - \lambda}.\frac{1}{\tilde{b}_0}[\ddot{A}(z^{-1})y(k) - \{\tilde{B}(z^{-1}) - b_0\}u(k-d)] \tag{B.10}$$

$$= \frac{\tilde{b}_0}{\tilde{b}_0 - \lambda}.\bar{u} \tag{B.11}$$

where

$$\bar{u} = \frac{1}{\tilde{b}_0}[\ddot{A}(z^{-1})y(k) - \{\tilde{B}(z^{-1}) - b_0\}u(k-d)] \tag{B.12}$$

In order to ensure proper setpoint following, even in the case of inaccuracies in identification, we adopt strategy II. In this case we assume the pseudo-output as

$$\bar{\phi}(k) = y(k) + \lambda[u(k-d) - u(k-d-1)] \tag{B.13}$$

Hence one equivalently gets

$$\ddot{A}(z^{-1})\bar{\phi}(k) = [B(z^{-1}) + \lambda(1-z^{-1})\ddot{A}(z^{-1})]u(k-d) \tag{B.14}$$

$$= \bar{B}u(k-d) \tag{B.15}$$

B.1.2 Nonlinear deterministic plant

In this case we assume that the plant model is given by

$$y(k) = f\left[y(k-d),\cdots,y(k-d-n_y);u(k-d),\cdots,u(k-d-n_u)\right] \quad \text{(B.16)}$$

Hence the inverse dynamics model is

$$u(k-d) = g[y(k),y(k-d),\cdots,y(k-d-n_y);u_a(k-d-1),\cdots, u(k-d-n_u)] \quad \text{(B.17)}$$

In the line of section 4.4, let

$$\phi(k) = Py(k) + \frac{\rho}{\hat{h}}u(k-d) \quad \text{(B.18)}$$

By backpropagating a one through the neural network $\hat{g}$ can be obtained. Using equation B.18 the pseudo-output can then be calculated as earlier. Hence the estimate of the control input given to the plant is given as

$$\hat{u}(k-d) = \tilde{g}[\phi(k),\phi(k-d),\cdots,\phi(k-d-n_y);u(k-d-1),\cdots, u(k-d-n_u)] \quad \text{(B.19)}$$

Neural Network Controller

Using the neural net controller we determine u(k-d). The inputs to the neural network being the past pseudo-outputs and the inputs as well as the desired value of the pseudo-output.Hence the equation of the controller is given by

$$u(k-d) = \tilde{g}\left[\phi_d,\cdots,\phi(k-d-n_y);u(k-d-1),\cdots,u(k-d-n_u)\right] \quad \text{(B.20)}$$

Corresponding to strategy I adopted in the case of linear nonminimum phase systems we obtain, with slight manipulations, an equation analogous to equation B.11

$$\bar{u}(k-d) = \tilde{g}[y_d, \phi(k-d-1), \cdots o(k-d-n_y); u(k-d-1), \cdots, u(k-d-n_u)] \quad \text{(B.21)}$$

$$\text{and} \quad u(k-d) = \frac{\bar{u}(k-d)}{1-\rho} \quad \text{(B.22)}$$

The error is estimated as the difference between $u(k-d)$ and $\hat{u}(k-d)$ as earlier. But as already pointed out, in the case of inaccuracies in the estimation of parameters, the above strategy may cause an offset between the actual output and the desired output.

In order to ensure setpoint following we use the strategy II as in the case of the linear systems. In this case we let the pseudo output be given by

$$o(k) = y(k) + \frac{\rho}{\hat{g}(.)}[u(k-d) - u(k-d-1)] \quad \text{(B.23)}$$

This strategy is equivalent to the addition of an integrator in the forward path as shown in figure 4.10. Hence in order to control an inverse unstable plant using strategy II, we add an integrator before the plant. The remaining scheme is similar to that used in the case of a deterministic inverse stable nonlinear plant.

B.2 Simulation results

B.2.1 Inverse unstable deterministic linear plants

A plant given by the following equation is considered.

$$y(k) - y(k-1) + 0.24y(k-2) = u(k-1) + 1.5u(k-2) \quad \text{(B.24)}$$

The poles of the plant are at 0.4 and 0.6 while the zero is at -1.5. Since the zeros of the plant are outside the unit circle, the plant is inverse unstable or nonminimum phase.

The control of nonminimum phase plants has been done using the concept of generalized minimum variance control. This results in costing of control too. The aim in this case is to minimize the variance of a pseudo output.

A single layered neural network with a unity linear activation unit is used to learn the pseudo-inverse dynamics of the plant. The desired set point is varied between +5 and -5 after every 50 iterations. The variable scalar learning rate is as given in equation A.16. The weights of the neural network are updated at every instant of time using the backpropagation algorithm.

Two different strategies of calculating the pseudo output are used. The control resulting from strategy I is shown in figure B.1. A problem which arises in this case is that the input signal may become unbounded due to a poor estimation of $\tilde{b}_0$ which is in the denominator of equation B.11. To overcome this problem a heuristic proposed in [40] is used. This consists of replacing $1/\tilde{b}_0$ with $\tilde{b}_0/(\tilde{b}_0^2+p)$. An appropriate value of p is to be found through simulations. In our case it has been found to be about 0.000185. The resulting strategy does take quite some time to tune. Costing is put on control. Because of the costing of control, the input amplitude is also relatively small. The input amplitude is related inversely to λ, the costing on control. In order to ensure setpoint following strategy II has been applied. The resulting control is shown in figure B.2. It can be seen from the figure that setpoint following is quite good. This is as expected since as pointed out earlier this strategy is equivalent to the addition of an integrator ahead of the plant.

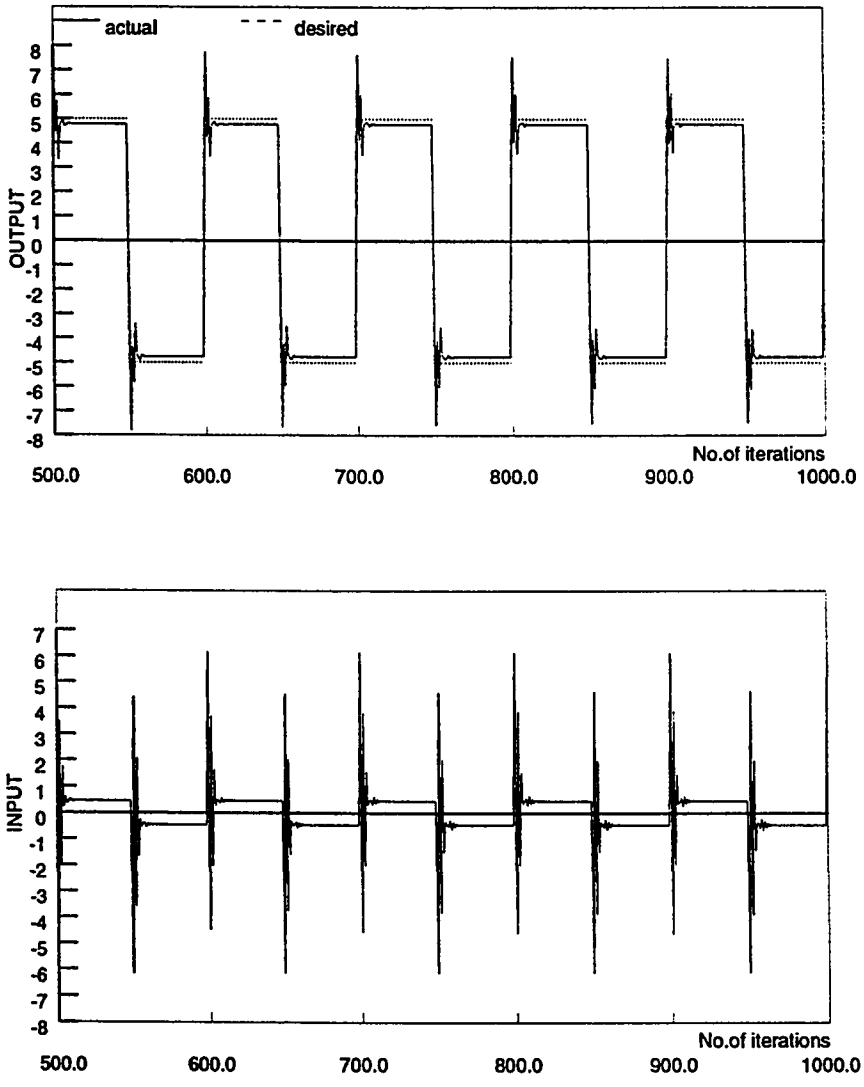


Figure B.1: Deterministic linear plant using the GMV strategy I

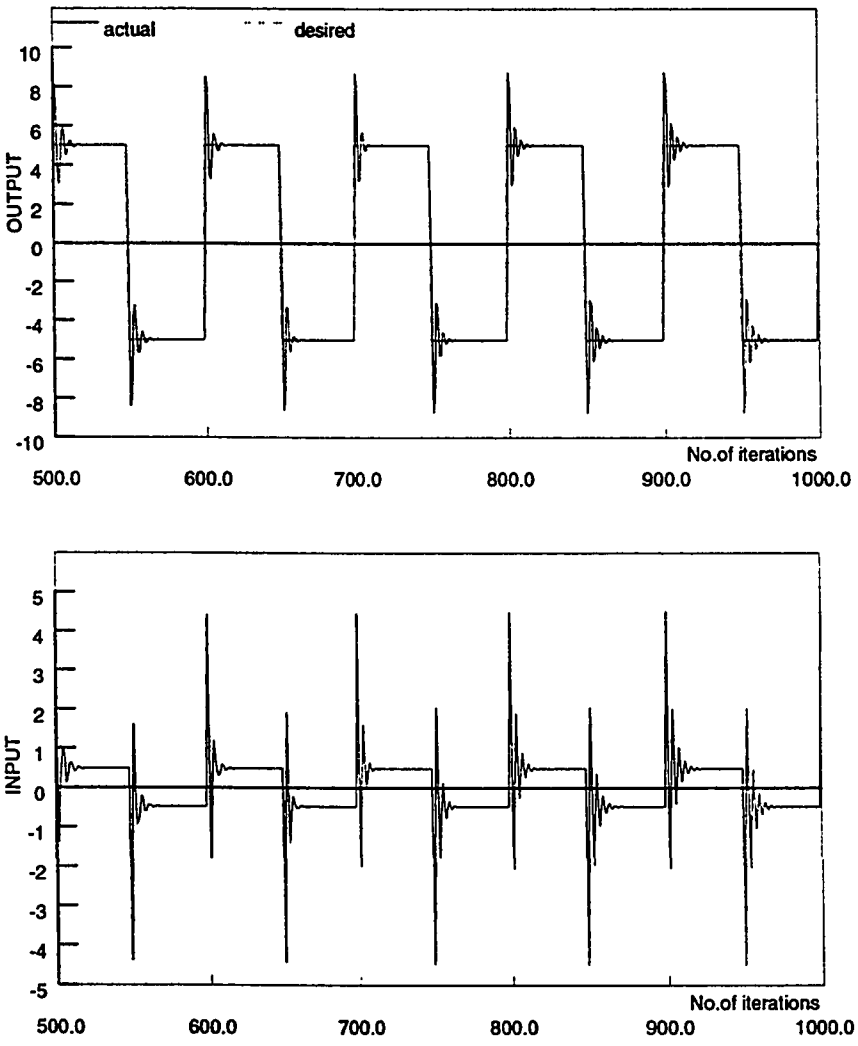


Figure B.2: Deterministic linear plant using the GMV strategy II

The value of λ used in the above simulations is set to 1. A high value of λ reduces the control input. A low feasible value of λ causes the opposite effects. This was also verified by simulation.

B.2.2 Deterministic nonlinear plants

In this example. the linear part of example 2 is modified to get an inverse unstable plant. The plant has been given by

$$y(t) - 1.6y(t-1) + 0.63y(t-2) = u'(t-1) + 1.5u'(t-2) \qquad \text{(B.25)}$$

with

$$u'(t) = \begin{cases} 0 & \text{if } \mid u(t) \mid \leq 0.1 \\ u(t) - 0.1 \text{ sign } \{u(t)\} & \text{otherwise} \end{cases}$$

The linear part of the plant has a zero at -1.5. When the minimum variance strategy is applied to this plant,the algorithm blew up as expected. As pointed out in chapter 4. the control of nonminimum phase plants is done using the concept of generalized minimum variance control. But in the case of a nonlinear system, the weighting has to be proportional to $\partial u(k-d)/\partial\phi(k)$. Hence this function is also needed to be identified.

Dither has been added for the first 1000 iterations to the input. The setpoint is varied between 1 and -1 after every 50 iterations. Results using both strategy I and strategy II have been provided in figure B.3. It can be seen that in both cases the performance of the plant is satisfactory. As pointed out earlier, namely that in the case of inverse unstable deterministic linear plants, the heuristics proposed should be used to ensure that the input is within limits. To avoid any singularity problems, we approximate $1/g$ as $g/(g^2+p)$. This strategy does control the plant effectively as

can be seen from figure B.3. It should however be pointed out that there may be an offset associated with this strategy. This is due to the feedforward gain being used. The offset has also been found to depend on p in the term $g/(g^2+p)$. A suitable value of p depends on the model being simulated and in our case, it was found to be about 0.001.

To overcome the problem of the offset, we have also used the second strategy as proposed in chapter 4. The simulation results are shown in figure B.4. It can be seen that the offset is overcome using this strategy. Further, the variations in the controller input are also small as expected. The weighting factor in both of the above simulations has been taken to be 0.25. Further reductions in the input magnitudes or variations can be obtained by increasing the costing value, i.e. by increasing ρ.

Another problem which has been faced is that, at times the learning by the neural network became stagnant for many iterations. This has been diagnosed as being due to locking. In order to overcome this, a small constant has been added to the derivative of the neuron activation function for some iterations. This is equivalent to replacing $(1-Y_k)(1+Y_k)$ in equation 3.14 by $(1-Y_k)(1+Y_k)+a$ where a is the small constant added. Obviously, this has to be removed after some time, otherwise the true derivative will not be used in the backpropagation method. In the simulation, this constant is kept as 0.01 for the initial 500 iterations.

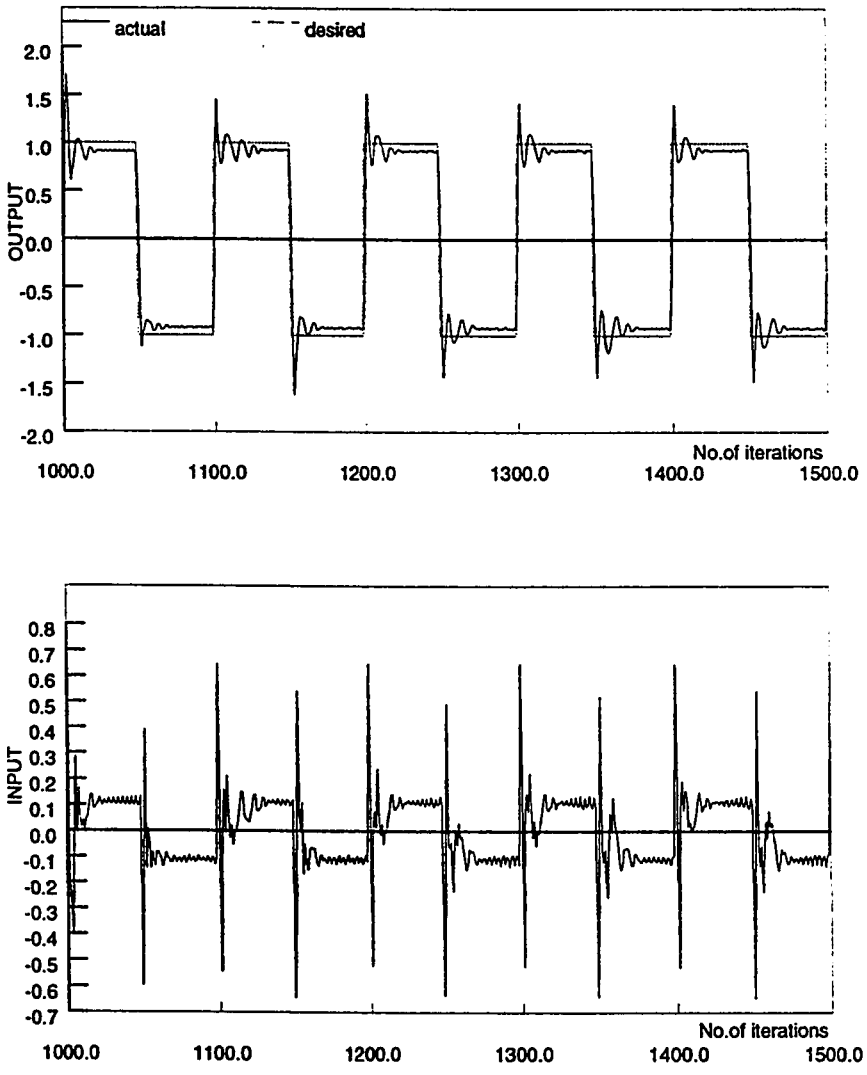


Figure B.3: GMV control of a deterministic inverse unstable plant with feedforward adaptation

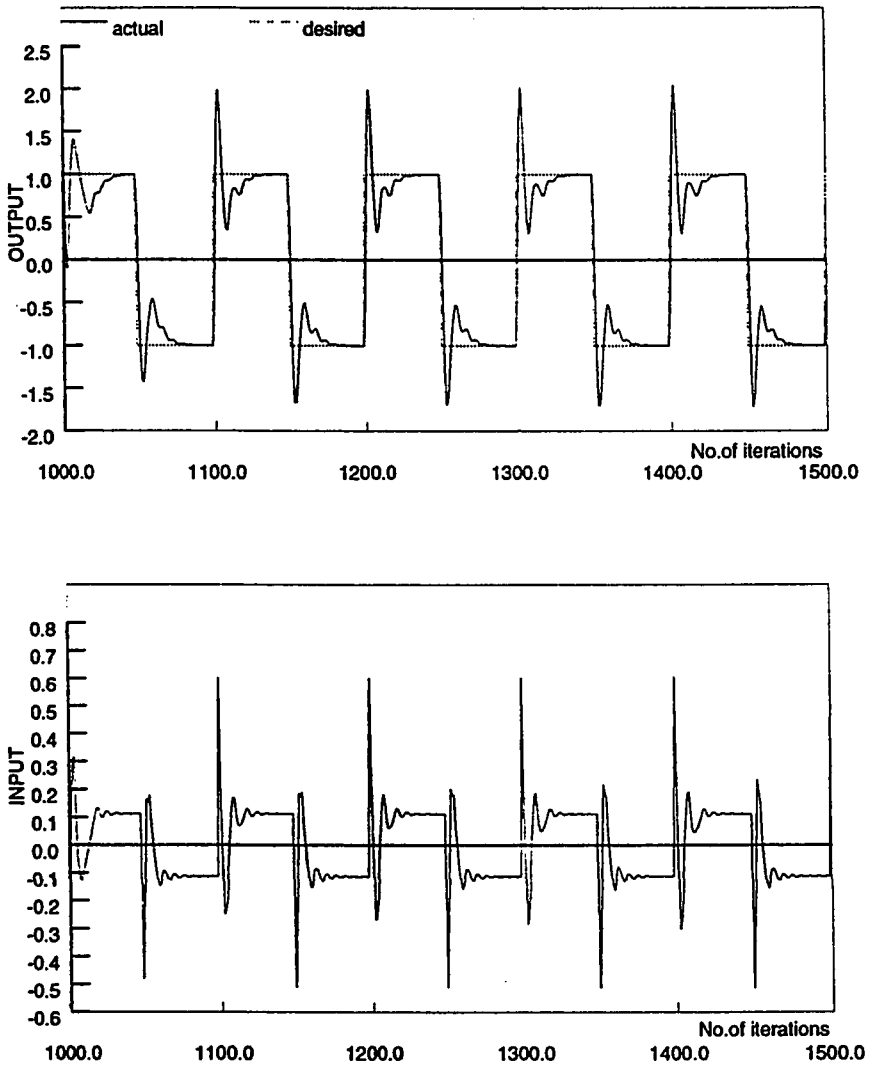


Figure B.4: GMV control of a deterministic inverse unstable plant using an integrator

Appendix C

C.1 Variable control costing

The pseudo output is given as

$$\phi(k) = y(k) + \lambda u(k-d) \tag{C.1}$$

This appendix shows that while using the generalized minimum variance self-tuning algorithm proposed in this paper, the control weighting parameter λ in (C.1) must be chosen to be inversely proportional to $\partial^+ u(k-d)/\partial\phi(k)$. The need of a variable control weighting in nonlinear plant was first pointed out in [41] and was further confirmed by Sales and Billings in [42].

The cost function minimized is given by

$$J = \frac{1}{2}E(\hat{e}^2(k-d)) = \frac{1}{2}E\|u(k-d) - \hat{u}(k-d)\|^2 \tag{C.2}$$

The object of the self tuning control is to find the new control input by minimizing J. The solution is obtained by setting $\partial J/\partial u(k-d) = 0$ which, together with (C.1), gives

$$E\left[(1 - \frac{\partial^+ \hat{u}(k-d)}{\partial\phi(k)}\lambda)\hat{e}(k)\right] = 0 \tag{C.3}$$

However, in the proposed approach, the control is calculated by setting $\hat{e}(k) = 0$ which assumes that $E\{\hat{e}(t)\} = 0$. The two schemes thus become equivalent if λ in (C.3) is inversely proportional to $\partial^+ u(t-d)/\partial\phi(t)$. In this case, it is ensured that the calculated control minimizes the variance of $\hat{e}(t)$. Note that in the linear case, the partial derivative is constant and therefore a variable control costing is not needed.

Bibliography

[1] Narendra K.S. Intelligent control. *IEEE Control Systems Magazine,*, pages 37–38, 1991.

[2] Kalman R.E. Design of self optimization control system. *Transaction of ASME* ,, 1959.

[3] Astròm K.J. Introduction to stochastic control theory. *Academic Press,Lund Institute*, 1970.

[4] Astròm K.J. and Wittenmark B. On self tuning regulators. *Automatica*, 9:185–199, 1973.

[5] Astròm K.J. and Wittenmark B. Self tuning controllers based on pole zero placement. *Proc. IEE.*, 127 Pt.D(3):120–130, 1980.

[6] Wellstead P.W. and Sanoff S.P. Extended self-tuning algorithms. *International Journal of Control*, 34(3):433–455, 1981.

[7] Clarke D.W. and Gawthrop P.J. Self tuning control. *Proc.IEE*, 126(6):633–640, 1979.

[8] Clarke D.W. and Gawthrop P.J. Implementation and application of microprocessor-based self-tuners. *Proc.IEE.*, pages 197–208, 1977.

[9] Clarke D.W. and Gawthrop P.J. Self tuning controller. *ProcIEE*, 122(9):929–934, 1975.

[10] Clarke D.W. and Gawthrop P.J. Some interpretations of the self tuning controller. *ProcIEE*, 124:889–894, 1977.

[11] Clarke D.W. Model following and pole placement self-tuners. *Optimal Control Applications and Methods*, 3:323–335, 1982.

[12] Clarke D.W. Self tuning control of non-minimum phase systems. *Automatica*, 20(5):501–517, 1984.

[13] Astrom and Wittenmark B. Practical issues in the implementation of self-tuning control. *Automatica*, 20(5):595–605, 1984.

[14] Fukuda T. Watanabe K. and Tzafestas S.G. An adaptive control for carma systems using linear neural networks. *International Journal of Control*, 56(2):483–497, 1992.

[15] Narendra K.S. and Parthasarathy K.. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[16] Chen F.C. Backpropagation neural networks for nonlinear self-tuning adaptive control. *IEEE Control systems magazine*, 10(2):44–48, 1990.

[17] Hunt K.J. et al. Neural networks for control systems–a survey. *Automatica*, 28(6):1083–1112, 1992.

[18] Narendra K.S. and Levin A.U. Control of nonlinear dynamical systems using neural networks :controllability and stabilization. *IEEE Transactions on Neural Networks*, 4(2):192–206, 1993.

[19] Yabuta T. and Yamada T. Neural network controller characteristics with regard to adaptive control. *IEEE Transactions on systems,man and cybernetics*, 22(1):170–176, 1992.

[20] Chen S. et al. Parallel recursive prediction error algorithm for training layered neural networks. *International Journal of Control*, 51(6):1215–1228, 1990.

[21] Ahmed M.S. Improved backpropagation algorithms for faster convergence submitted to. *IEEE Transactions on Neural Networks*, 1993.

[22] Narendra K.S and Mukhopadhyay S. Intelligent control using neural networks. *IEEE Control systems magazine*, 12(2):11–19, 1992.

[23] Barto A.G. Sutton R.S. and Williams R.J. Reinforcement learning is direct adaptive optimal control. *IEEE Control systems magazine*, 12(2):19–23, 1992.

[24] Cruz C. Understanding neural networks:a primer. *Cutter Information Corp.*, 1990.

[25] Wasserman P. Neural computing. *Van Nostrand Reinhold,New York*, 1988.

[26] Touretzky D.S. and Pomerleau D.A. What's hidden in the hidden layers. *BYTE,*, pages 227–233, 1989.

[27] Webros P.J. Backpropagation through time:what it does and how to do it. *Proc. of the IEEE*, 78:1550–1560, 1990.

[28] Irfan Haseeb. Computationally efficient self tuning algorithms. *M.S.Thesis,Dept.of Systems Engg,KFUPM*, 1989.

[29] Soderstom and P.Stoica. System identification. *Prentice Hall*, 1989.

[30] Clarke D.W. and Mohtadi C. Properties of generalized predictive control. *Automatica*, 25(6):859–875, 1989.

[31] Velde P.G.A. Keyser R.M.C. and Dumortier F.A.G. A comparative study of self-adaptive long range predictive control methods. *Automatica*, 24(2):149–163, 1988.

[32] Leontaritis and Billings S.A. Input-output parametric models for nonlinear systems. *International Journal of Control*, 41:303–344, 1985.

[33] Haaser and Sullivan J.A. Real analysis. *Van Nostrand Reinhold:New York*, 1971.

[34] Cybenko G. Approximation by superposition of sigmoidal function. *Mathematics of Control signals and systems*, 2(4):303–314, 1989.

[35] Stinchcomb M. Hornik K. and White H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[36] Werbos P.J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

[37] Su H. Qin S. and McAvoy T.J. Comparison of four neural net learning methods for dynamic system identification. *IEEE Transactions on Neural Networks*, 3(1):122–130, 1992.

[38] Psaltis et al. A multilayered neural network controller. *IEEE Control Systems Magazine*, pages 17–21, 1988.

[39] Ahmed M.S. and M. Farooq A. Neural net based direct self-tuning control using the gmv principle submitted to. *Automatica*, 1993.

[40] Astrom and Wittenmark. Adaptive control. *Addison Wesley Publishing Co.*, 1989.

[41] Anbumani K.Sarma I.K and Patnaik L.M. Self tuning minimum-variance control of nonlinear systems of the hammerstein model. *IEEE Transactions on Automatic Control*, pages 959–961, 1981.

[42] Sales K.R. and Billings S.A. Self-tuning control of non-linear armax models. *International Journal of Control*, 51:753–769, 1990.

[43] Ahmed M.S. Sequential gauss-newton algorithms for training of neural networks. *submitted to IEEE Transactions on Acoustics,Speech and Signal Processing.*

Vita

- M.M.Farooq Anjum
- Born in Hassan, India
- Received Bachelor's degree in Mechanical Engineering from the S.J. College of Engineering in November 1990.
- Worked as Mechanical Design Engineer at Pragati Tools Pvt. Ltd. Bangalore, India .
- Joined Systems Engg. Dept. at KFUPM in Spring 1992.
- Completed Master's degree requirements at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia in Fall 1993.